Architecture-Centric Traceability for Stakeholders: Technical Foundations

Hazeline U. Asuncion and Richard N. Taylor
ISR Technical Report # UCI-ISR-11-2
May 2011

Abstract— Software traceability, is recognized for its utility in many development activities. Achieving traceability in practice, however, is difficult because of the complex interaction between factors like high costs, heterogeneous artifacts and tools, and varied stakeholder interests. Architecture-Centric Traceability for Stakeholders (ACTS) is a technical framework that considers the economic and social challenges to traceability. This framework connects distributed and varied artifacts around concepts represented by the architecture, enables stakeholders to control the traceability capture via tool extensibility and customization, and prospectively captures links in the background as users perform their development tasks. We discuss open hypermedia and rules as the supporting mechanisms of our framework. We demonstrate the technical feasibility of our approach through a case study in software acquisition research and an exemplar implementation. We also discuss means of increasing the practicability of our framework based on user feedback.

Index Terms— Documentation, Hypertext/Hypermedia, Software Architectures, Software Traceability

1 Introduction

Software development is a complex process that rests upon creating, arranging, juxtaposing, analyzing, and transforming information artifacts. These myriad heterogeneous artifacts are interrelated in numerous ways. Recognizing, maintaining, and using these relationships is fundamental to development and subsequent system evolution. Too often, however, these relationships are only "maintained" in the minds of software engineers and eventually forgotten. The isolation of artifacts by tools, development teams, and geographic locations causes related information to drift apart, leading to obsolescence and inconsistencies between these information units. Software traceability aims to cross these barriers to explicitly connect related information artifacts. When successful, software traceability facilitates system comprehension, impact analysis, system debugging, and communication between stakeholders [48, 100, 76, 103, 105]. Not only does traceability support software development by making relevant artifacts accessible to all members of the development team, but it also lowers the cost of software maintenance [103] – at a minimum by speeding up access to the information needed when making changes.

Despite these benefits, the lack of effective traceability, which we call the traceability problem, is a long-standing shortcoming in software development [64]. Many approaches have proven infeasible in practice [9, 25, 110]. Manual techniques for establishing and maintaining traceability links are tedious and error-prone. Consequently, software engineers generally view traceability obligations as additional imposed work with no direct benefits [76]. In addition, automated techniques often require human intervention [70]. Thus, high overhead remains an issue.

We posit that these limitations stem from a narrow understanding of the traceability problem. Our survey of reported difficulties with traceability reveals that many interacting factors hinder effective traceability, including high costs [76, 104], complex interrelationships between artifacts [9, 13], heterogeneity of artifacts and tools [48, 82], and varied stakeholder interests [64, 104, 120]. These factors, which reflect economic, technical and social perspectives, must all be addressed to realize the benefits of traceability.

- *H.U. Asuncion is with the Computing and Software Systems, University of Washington, Bothell, Box 358534, 18115 Campus Way NE, Bothell, WA 98011-8246. E-mail: hazeline@u.washington.edu*
- *R.N. Taylor is with the Institute for Software Research, University of California, Irvine, ICS2-203, Irvine, CA 92697-3455 . E-mail: taylor@ics.uci.edu*

We recognize the complexity of the traceability problem, and thus, the purpose of this paper is to lay the technical foundations for building a highly customizable traceability tool that can integrate current techniques while considering the economic and social perspectives. We demonstrate the technical feasibility of our approach through a case study in software acquisition research and an exemplar implementation. We also provide user feedback regarding the usage of a tool built upon these foundations. The contributions of our approach, Architecture-Centric Traceability for Stakeholders (ACTS) are as follows:

- A means of connecting distributed and varied artifacts by linking them to concepts represented by the *architecture*
- The ability of *stakeholders* to *control* traceability capture via tool extensibility and customization features
- The *prospective* capture of links in the background, as software engineers perform development tasks
- The ability to automatically capture links to *heterogeneously* represented artifacts such as graphic files, presentation files, and various media files
- The *decoupling* of heuristics for capturing links from the underlying capture mechanisms, facilitating the adaptability of the tool to different contexts.

Our previous work discusses the similarities between traceability and data provenance in e-Science and introduces possibilities for leveraging e-Science techniques in software engineering [26]. Our more recent work, which combines ACTS with a machine learning technique known as topic modeling, demonstrates the feasibility of integrating existing information retrieval techniques with the ACTS framework [24]. While that work focuses on the semantic analysis and visualization of software artifacts within a traceability context, this work focuses on the core mechanisms of prospective traceability, which are open hypermedia techniques and rules.

The rest of the paper is organized as follows. The next section briefly analyzes the problem and presents current approaches. Section 3 presents ACTS and the main elements of our framework. We demonstrate the technical feasibility of architecture-centric links in a case study of software acquisition research in Section 4. We then demonstrate the technical feasibility of prospectively capturing user customized links across heterogeneous artifacts through an exemplar implementation in Section 5. Section 6 provides user feedback. We conclude with open research areas and future work.

2 Problem Analysis & Existing Techniques

Hindering factors to traceability stem from economic, technical, and social perspectives. An interplay also occurs between these perspectives such that factors from one perspective affects or is affected by factors from another [25]. This section presents our survey of these challenges and how current approaches address them [23].

2.1 Economic Perspective

The economic perspective focuses on the cost of supporting traceability. Capturing and maintaining traceability links incurs high costs in terms of labor hours [7, 103, 76, 81], and high cost is one of the major hindering factors to traceability [54, 76]. A case study of a large government-funded project reports that the costs of implementing traceability is more than double the normal documentation costs [103]. Some practitioners argue that time spent in traceability tasks could have been allocated to writing software code [28, 36]. Even with companies that are willing to pay the high costs of traceability, the expected benefits are often not realized [112, 104]. Other sources of costs include increased documentation, purchase or development of a trace tool, and user training [25, 103].

To mitigate the cost, some approaches examine the tradeoffs between cost and quality [52] or between cost and benefit [53]. Regarding cost and quality tradeoff, one can attempt to reduce the level of granularity of traces in order to save costs while still maintaining an acceptable level of link quality [52]. For instance, it has been shown that tracing at the method source code level is more expensive than tracing at the class source code level even though there is usually not much difference in the quality of information obtained. Meanwhile, approaches that examine the cost-benefit tradeoff concentrate on tracing only higher value links in order to minimize cost [53]. This

scheme may require a cost-benefit analysis for each project. A similar approach allocates more trace support to the crucial parts of the software system [40]. Another technique is to take a minimalist approach and only capture links that are important to a class of users, such as developers [36].

2.2 Technical Perspective

While the economic perspective examines the costs in supporting traceability, the technical perspective looks at the complexity of tracing due to the heterogeneity of artifacts, the heterogeneity of tools, the combinatorial explosion of the artifact space, and the continuous and independent evolution of artifacts.

2.2.1 Heterogeneity of Artifacts

The heterogeneity of artifacts is a factor that contributes to the traceability problem. Artifacts produced in the course of software development vary in their levels of formality, ranging from unstructured documents to highly formal code. The differing formats and notations by which artifacts are expressed as well as the different levels of abstractions represented by the artifacts present challenges to establishing traces across different artifact types [69].

Approaches that address the heterogeneity of artifacts include natural language processing, two-dimensional traceability, model transformation, and artifact translation. Natural language processing techniques use language structure to determine links between various text artifacts [27, 32, 109]. Two-dimensional traceability enables tracing between information within an artifact (vertical traceability) and information across different types of artifacts (horizontal traceability) [82]. Another approach, model transformation, enables tracing design artifacts across different levels of abstraction. Transformations can vary in the level of automation. Fully automated transformation use a transformation specification on a design artifact to produce a realization that is at a lower level of abstraction [10, 77, 89, 105]. Still another approach translates heterogeneous artifacts into a common format in a repository. Trace relationships between artifacts are automatically generated within the repository [13, 39]. These approaches are generally limited to text-based artifacts.

2.2.2 Heterogeneity of Tools

Tracing software artifacts across different tools is also difficult due to the lack of interoperability between tools [48, 65, 103]. The separation of information by tools is known as the “islands of information” problem [13]. Changing the artifacts outside a trace tool does not guarantee that the artifacts inside the trace tool are updated [48, 39]. The lack of interoperability between different tools necessitates redundant data entry [20] which adds to the overhead of reconciling data [13].

One way to address tool heterogeneity is through the use of a shared repository and specialized code. For instance, different tools can exchange data via a shared repository [25]. The tools communicate with the shared repository via customized code. This approach avoids the problem of redundant data entry since artifact changes are always reflected in a shared data repository. A similar approach is to use a shared data model and a communication channel where the subscribed tools, such as a browser and an IDE, listen for published updates to the data model [63]. When a matching criterion is found between the code in the IDE and the resource on the browser, the tool prompts the user whether to link the visited resource to the source code.

Another approach is to use a shared repository with tool monitors and pre-defined heuristics [42]. The shared repository stores the metadata of newly generated or modified artifacts. Tool monitors, invoked by an Update Module, track any changed artifacts from various sources (e.g. Bugzilla, CVS, mailing list archive) and updates the shared repository. Links are generated by indexing the artifacts and applying pre-defined heuristics within the tool. These techniques, however, do not have extensible mechanisms by which users can integrate their own tools into the traceability environment.

Another way to tackle tool heterogeneity is through the use of open hypermedia concepts. Open hypermedia adapters can be used to manually create links across tools boundaries [15]. Later in the paper, we will show how we build upon open hypermedia concepts to automatically capture trace links.

2.2.3 Explosion of the Artifact Space

Tracing across various artifacts in a software development lifecycle is difficult due to the sheer number of artifacts and the complex relationships between these artifacts [13]. Capturing an insufficient number of trace links can have negative effects such as lower system quality and increased project costs and time [37, 100]. On the other hand, capturing too many traces is unwanted. Excessive traceability is known to be unmanageable [37, 48] and can negatively impact the accuracy of links [52]. Thus, it is important to know the boundary between adequate and excessive tracing.

There are different ways of bounding the problem space of artifacts and artifact-relationships. The agile community advocates a lean traceability approach where only relevant traces to the developers are captured [36]. The selection of specific artifacts to trace can also be based on the project manager's discretion or the information gleaned from past projects [48]. These approaches require a basic understanding of the system or previous experience.

Another approach is recovering candidate trace links automatically through information retrieval (IR) techniques. To date, trace recovery techniques have not been able to provide fully accurate links [39, 68]. Captured traces may only be as accurate as the user input [52]. One requirement of these techniques is artifact preprocessing [35]. Even with sophisticated IR techniques, it is difficult to achieve high recall and precision rates, where recall is defined as the percentage of retrieved links out of all relevant links and precision is the percentage of correct links out of the retrieved links [36].

2.2.4 Maintenance of Traceability Links

Maintaining trace links is also challenging. Links quickly deteriorate because artifacts change continuously and independently and the changes are not reflected in the related artifacts [62]. For example, changing a requirement necessitates the update of all the corresponding links and related artifacts. Without a systematic update approach, the cost of maintaining traceability can be very high. There is also no guarantee that all the impacted links are updated.

Techniques to maintain trace links include processes to control artifact changes. Artifact changes can be controlled by establishing a development process to disallow software engineers from directly changing artifacts. One such technique requires changes to be approved by a review board [108]. Since this process imposes high overhead, only high visibility documents go through review boards. Another technique is providing a downstream development team a process whereby they can control changes to the requirements made by an upstream functional development team [20].

Changes can also be automatically cascaded between tools or artifacts [25, 37, 1, 121]. For instance, Rational RequisitePro automatically updates its data when the related Word documents change [1]. Embedding information objects, which are automatically updated, into various documents is another means of cascading changes across different artifacts [121]. Moreover, artifact changes can be cascaded through event-based traceability (EBT). EBT uses a publish-subscribe mechanism to relate various artifacts to the requirements artifact [37]. Thus, when a requirements artifact changes, the subscribed artifacts are notified. This approach requires a manual registration of artifacts with the requirements artifact.

Still another approach is to manage the relationships of all artifact types within a tool so that changes can automatically be reflected in the links [97]. This requires a pre-specification of the artifact types and their relationships.

2.3 Social Perspective

The social perspective is equally important, since it focuses on the interaction of stakeholders with traceability, such as differing expectations, low motivation, and lack of artifact visibility. It is recognized that people play a crucial part in determining the quality of traceability links [20, 68, 70].

2.3.1 Different Expectations of Traceability Tool

Implementing software traceability is difficult since traceability has different meanings to different people [65]. Consequently, stakeholders have different expectations of a trace tool [103]. For instance, a maintenance engineer expects support for impact analysis while a project manager expects support for tracking project status. One way to address different stakeholder expectations is by identifying the key users of a trace tool and developing custom in-house extensions to existing trace

tools [104] or developing custom trace tools for an organization [25].

2.3.2 Low Motivation for Performing Traceability Tasks

In general, software engineers have little or no motivation to perform traceability tasks [28, 76]. To them, traceability tasks are "laborious" [10] and "burdensome" [9]. In one study, half of the subjects who were commissioned to verify trace links dropped out because they "disliked" tracing [68]. There are several reasons for the low motivation of software engineers. One reason is that traceability tasks are additional imposed work with no direct benefits [20, 76], known as the Traceability Benefit Problem [20]. Other reasons include the lack of understanding of the usage of trace information and the lack of first-hand knowledge of the artifacts [20].

One way to address the low motivation for performing traceability tasks is by coupling traceability tasks with the development process [60, 99] or with traceability usage [20]. This approach may require high overhead in setting up the process capture to automatically determine trace links [60, 99]. Another method is to use trace information to support stakeholders in their development tasks [22, 25] thereby providing direct benefits to them.

2.3.3 Lack of Artifact Visibility

Traceability across artifacts owned by different groups is difficult due to the lack of visibility of artifacts to those outside the groups. For example, the lack of visibility to the requirements' sources, which could be distributed among multiple groups, has been the most frequently cited problem by practitioners [65]. In addition, distributing the ownership of requirements among different groups makes it difficult to trace the dependency relationships among the requirements [104]. Lack of communication between groups is one of the factors that contribute to the lack of accessibility of artifacts [65].

Approaches that address the lack of visibility to artifacts include negotiating changes to upstream artifacts and publishing artifacts to a portal. Encouraging teams to negotiate the changes to upstream artifacts like requirements enhances communication between groups and increases the accessibility of artifacts [20]. Publishing artifacts to a portal raises the visibility and accessibility of artifacts to other groups [25].

2.4 Perspectives Interplay

The economic, technical, and social perspectives are highly intertwined. A factor in one perspective affects factors in others (see Figure 1). We provide some examples of the interplay between the different perspectives and explain why focusing on one perspective falls short of addressing the traceability problem.

There is a bidirectional relationship between the economic and technical perspectives. The economic perspective is a major factor in determining whether a traceability approach will be adopted by an organization [25]. The cost of establishing and maintaining trace links affects the number of artifact and the relationship types that will be traced by an organization. Since fine-grained tracing is more costly [29, 20], the economic perspective also determines the level of granularity that an organization is willing to trace. The technical perspective also affects the economic perspective. The level of tool support in establishing or defining traceability links heavily determines the cost of tracing [82]. The lack of interoperability between tools also contributes to the high cost of traceability since this necessitates redundant data entry and manual reconciliation [108, 13, 25, 103]. Moreover, there is a tension between capturing all possibly relevant links to ensure that no loss of knowledge occurs [48, 104] and taking a minimalist approach in trace capture [36] to lower the cost. There are currently no cost-benefit models that can guide organizations in selecting the types of artifacts, the level of granularity, and the types of relationships to trace [69].

A bidirectional interaction between the social and technical perspectives also exists. For instance, due to the low motivation of software engineers in performing traceability tasks, the captured traces were deemed to be unusable in one reported case study [20]. In addition, users have differing expectations [65], making it difficult to use an off-the-shelf trace tool without customization. The technical perspective affects the social perspective as well. If a trace tool supports the development activities of stakeholders, it is more likely to be adopted [25, 90].

Similarly, interactions occur between the economic and the social perspectives. Due to the high costs required in performing traceability tasks, most software engineers have an aversion toward traceability [76, 68]. The high startup and maintenance cost of the manual approaches is also one of the common complaints of developers [28]. Furthermore, lack of accessibility of artifacts between groups can make tracing across groups more costly since more time is spent locating artifacts.

There is also interplay between the three perspectives. One example of this interplay is the automation of trace link generation (see Figure 1). To mitigate the costs, information retrieval methods can be used to provide automated support for traceability [84, 17, 39, 71], at the risk of potentially establishing inaccurate links. To compensate for this technical shortcoming, human involvement becomes necessary. The generated candidate links must be post-processed by a human analyst [68, 39]. However, the economic difficulty of cost shows up again if the IR technique produces very large numbers of candidate links to be manually analyzed [81].

Another example of the interplay between the three perspectives is in the heterogeneous nature of the artifacts. Because it is difficult to automate data conversion between heterogeneous artifacts [112], these artifacts tend to be linked manually. Manual linking is a human intensive effort that is often viewed by developers as a burden. Thus, the technical difficulty of linking heterogeneous artifacts results to added economic overhead. This in turn then leads to developers' aversion to tracing, a social issue [28].

3 Technical Foundation for Adaptable Traceability: ACTS

To begin to address the multi-faceted traceability problem, we present Architecture-Centric Traceability for Stakeholders (ACTS), a technical framework that considers both the economic and social perspectives. The key elements of ACTS are centering links to the architecture, enabling stakeholders to control the link capture and usage, and capturing links prospectively. We detail how traceability link capture is enhanced by exploiting the information stored in the architecture as well as exploiting the stakeholder knowledge in how artifacts are related. ACTS is also undergirded by the supporting mechanisms of open hypermedia and rules.

3.1 Architecture-Centric Traceability

The first element of our approach, architecture-centric traceability, links all the artifacts to the architecture. Previous approaches have suggested centering links to the requirements [70] or to the code [52]. While there are advantages and disadvantages to both models, we posit that architecture-centric traceability provide advantages that

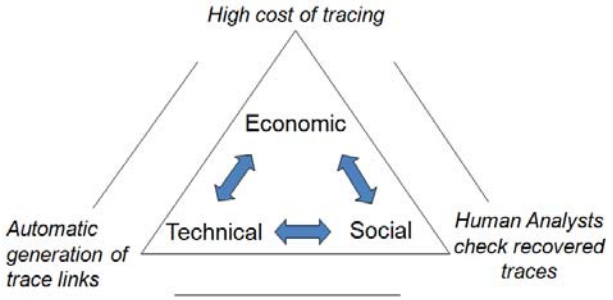

Fig 1: An example of an interplay between hindering factors from the three perspectives

are lacking in these previous model, and is worth one's consideration. To aid the reader, we first provide a brief background on software architecture research and recent advances in the field that support some of software traceability goals. We then move to discuss the rationale for using the model, how current approaches fall short of supporting architecture-centric traceability, and finally discuss contexts where the model may be less relevant.

3.1.1 Background in Software Architecture Research

While techniques have been proposed to capture links to the architecture, grounding all the links in the architecture is a novel concept, and one that proceeds from architecture-centric development [115]. Architecture-centric software engineering [31, 59, 67] conceptualizes software development activities with the system's architecture as a central focal point. We define software architecture broadly, as the set of principal design decisions about a software system [115]. This definition implies that every software system has an architecture—some set of design decisions that were made in its development. Principal design decisions can be expressed as the system's structure, functional behavior, interaction, and non-functional properties; this paper focuses on deci-

sions as expressed in the system's structure, e.g., the architectural style, the functional units, the mode of communication between the functional units, and the configuration of these units. These principal design decisions, as represented by the system's structure, provide product-based concepts by which traceability can be reliably anchored and supported.

Software architecture and architecture-based evolution research has provided techniques in modeling and analyzing software as well as in developing and evolving the code. Architecture description languages (ADLs) [85] have been used to model and analyze the software structure, behavior, distribution, and concurrency. Most of these approaches focused on discrete units of computation (components), discrete units of communication (connectors) and the arrangements of these units (configurations). Recent work in software architecture [30, 56, 92] resulted in extensible and modular ADLs and toolsets (e.g. Acme [58], ADML [111], and xADL 2.0 [44]). We use xADL which has mature tool support with respect to extensibility. We posit that centering links on the software architecture strikes a balance between rigor, formality, and level of abstraction. ADLs tend to have more rigorous syntax than most requirements capture languages (including natural languages) and are easily tied to other concrete artifacts such as components, connectors, their implementations, test cases, and so on. Thus, versionable architectural models are a plausible, stable form upon which to anchor traceability links. Extensible ADLs, moreover, can be expanded to capture stakeholder concerns such as modeling security, distributed systems, and product lines [43].

3.1.2 Rationale for Architecture-Centric Traceability

This section discusses the insights for centering links to the architecture.

Insight #1: There is an inherent relationship between the architecture and other artifacts in software development. The architecture, as represented by the system's structure, provides a central 'hub' through which artifacts can be coordinated (see Figure 2). Architectural models of the system's structure serve as a direct basis for activities such as implementation and testing, as well as future evolution of the system. Similarly, the architecture can serve as the primary connection of the software product to its requirements (if they exist), identifying and documenting the design decisions that are responsible for realizing those requirements. Thus, design decisions represented by the architecture, impacts, or is impacted by, all other development activities. We now discuss these relationships in detail.

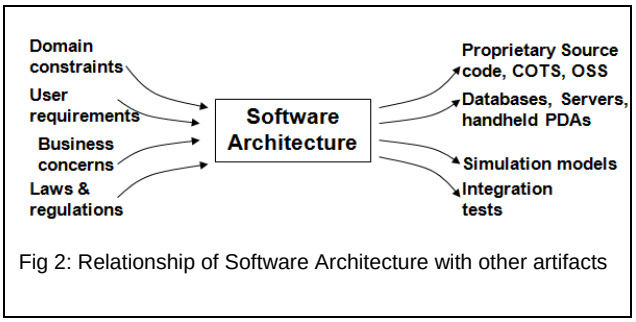

Fig 2: Relationship of Software Architecture with other artifacts

First, there is a direct relationship between requirements and architecture. Nuseibeh's Twin Peaks Model indicates that architecture plays an important role in the development and refinement of requirements [91]. In fact, Nuseibeh suggests that the concurrent development and evolution of both requirements and architecture, as observed in most software industry projects, is an effective means of developing software systems quickly. Taylor et. al. goes further in stating that the architecture provides the language whereby user needs can be concretely expressed in the requirements [115]. Moreover, software architectures have also been used to address non-functional requirements early in the software development lifecycle [115, 124].

Software architecture has also been used to assist developing and evolving implementations [16, 34, 95, 94] with tool support. The architecture serves to guide the implementation activity, ensuring that all design decisions are transferred to the code [115]. Research in software architecture has made strides in strengthening the relationship between the architecture and code [8, 45] through the use of architectural styles, frameworks, explicit implementation mappings, and code generation [115]. All these approaches provide an easy basis for establishing links, either generative or explicit, between architecture and implementation.

Not only does the architecture influence both the requirements and code, but it also has ties to test artifacts. Architecture-based testing enables a

system to be partially tested before it is actually implemented in the code [75, 124], allowing for the detection of errors early in the software lifecycle. Then as development progresses, architecture-based testing is extended to test the source code [119]. Given this influence that the architecture exerts on other artifacts, centering traceability links on the architecture certainly seems to be a reasonable choice.

Insight #2: The architecture itself contains information that aids in understanding the system and its related artifacts. The software architecture provides a comprehensive view of the system, enabling engineers to better understand the system in its entirety as well as in its individual computational units. Indeed, in large complex software systems, such as the Web, the software architecture is the only adequate guide to understanding the whole system [115]. Understanding this interconnection is particularly important in systems composed of heterogeneous subsystems such as legacy systems, open source software, and in-house proprietary software. In these types of systems where components may be black boxes, code-centric traceability no longer becomes a viable option. Finally, a better understanding of the entire system helps in identifying how an artifact may be related to the system.

Understanding the entire system also aids in understanding the traceability links across different system versions and across software product lines [72]. Links centered on the architecture can be updated to reflect system evolution. In addition, it is also more intuitive to trace links to the architecture when capturing links across product lines. Product line architectures (PLAs) have been the predominant means of evolving a product line and its individual products [72]. PLAs use well-understood constructs such as core, optional, and variant elements that aid in understanding the links to artifacts and which place the understanding of the artifact within the context of a product line.

In addition, the architecture contains the interconnections of the system, or its configuration, which can be used to infer links between other artifacts. For instance, a hierarchical structure in an architecture, as represented by a component containing subcomponents, may also indicate a hierarchical relationship between artifacts linked to the parent component and artifacts linked to the subcomponent. Similarly artifact links to directly connected components may imply relationships between those artifacts as well.

Insight #3: Architecture-centric traceability enables more efficient linking of select concepts as compared to requirements-centric or code-centric approaches.

The architecture has a well-defined structure: units of computation (or units of functionality), units of communication, and the composition of these units as a system. The architecture itself, if properly designed, has clearly defined non-overlapping concepts that are represented by these individual architectural units. Relating artifacts to these concepts simply means creating links between an artifact and an element or group of elements in the architecture. For example, individual requirements can be directly linked to the component or components in an architecture.

Meanwhile, these concepts are not immediately apparent in the requirements document. Concepts of functionality may be stated at a high level with vague notions of how they may be realized. Indeed, concepts of computation or communication may not actually be clearly stated since the requirements document usually concern itself with concepts in the problem space and not the solution space, as the architecture does.

Code-centric traceability also presents challenges. While the concepts of computation and communication are present in code, it is often difficult to identify these higher level conceptual boundaries at the code level. Thus, it is easy to capture more links than are necessary [52].

3.1.3 State of the Art in Linking to the Architecture

Techniques have been proposed to capture links to the architecture. For example, design rationale techniques link the design to the rationale to support reflection, communication among stakeholders, and analysis of past decisions [74]. There are numerous approaches and tools [80, 83, 33] for capturing and managing design rationale, but these are difficult to implement in practice [74, 114]. Another technique is use of model driven development (MDD) techniques to capture links across different design models. This approach tends to be limited to linking between two adjacent artifacts—the source and the target models [77].

Meanwhile, any artifact may be linked to the architecture regardless of its representation or level of formality. In addition, since the architecture can be modeled at different levels of abstraction, linking at these different levels of abstraction is also supported. Finally, artifacts may be linked to the architecture at any stage of its development, regardless of whether it is complete or not. As a result, a flexible traceability scheme can be adopted by centering links on the architecture.

3.1.4 Limitations of Architecture-Centric Links

While architecture-centric links provide advantages that are not provided by requirements-centric or code-centric traceability, there are instances when the architecture-centric model is not applicable. For instance, it might be necessary to capture links between a pre-defined set of artifacts, such as between requirements and test cases. There are also cases when links must be captured early in the software development lifecycle, while the problem domain is still being studied by the development team and the architecture has not been created. In addition, systems may not have a documented architecture and it may be too expensive or infeasible to recover an accurate system architecture. In these cases a placeholder, or "null" architectural model could be used until such time as a substantive model is developed.

3.2 Stakeholder-driven Traceability

The second key element is empowering stakeholders to both capture and use the traceability links they capture, which we call stakeholder-driven traceability. Oftentimes, the stakeholder who captures the links is not the same as the stakeholder who uses the links. We believe, however, that it is imperative for stakeholders who captured the links to also be able to use the links. We define stakeholders as individuals who are involved in traceability tasks or who have a vested interest in capturing relationships between software artifacts. This section provides a brief background to the role of humans in traceability, the rationale for tool-supported stakeholder-driven traceability, current tool support for stakeholder-driven traceability, and contexts where the model is less relevant.

3.2.1 Background in the Role of Humans in Traceability

As mentioned earlier, researchers recognize that humans play a critical role in the adoption and success of a traceability approach [70, 25, 103, 20]. Stakeholder-driven traceability is not a new idea [20, 90, 25], but it has generally been constrained by limited tool support [120]. This section examines the human perception of traceability and contexts where traceability has been implemented with some measure of success.

Despite the acknowledged need for traceability [62, 55], many practitioners have an aversion to traceability [10, 9]. There are several reasons. To many software engineers, traceability means manually creating traceability links which quickly deteriorate and becomes unusable [62]. Even with some automated support, traceability to some is a laborious, time-consuming task of identifying the correct links among the candidate links [70]. Still others view traceability as redundant data entry across different tools [25, 20]. Others, however, may want to capture traceability links to support their development tasks, but are unable due to the lack of adequate tool support [120].

There are a few instances where stakeholders adopted a traceability strategy with some level of success [21, 90, 25]. In one small development group, traceability links helped the engineers to coordinate and control changes to requirements and to identify which artifacts could be reused [21]. In another setting, developers used traceability links to increase program comprehension, to avoid architectural erosion, and to support change impact analysis [90]. Still in another setting, software engineers created traceability links to track project status, to support requirements analysis and to support high-level design [25]. A common thread that runs across these different contexts is that the stakeholders chose the artifacts to link and they used the captured links to support their software development tasks.

3.2.2 Rationale for Tool-supported Stakeholder-driven Traceability

Based on this observation, we conclude that an effective traceability scheme should provide tool support to enable stakeholders to control the capture of traceability links and to use the captured links to support their software development tasks. We discuss the implications for providing tool support for stakeholder-driven traceability.

Insight #1: An effective traceability tool must cater to varied stakeholder interests in capturing traceability links. Traceability has different connotations to different stakeholders [64]. Even within a given project, stakeholder interests in traceability vary [103, 25]. This variability occurs in the types of artifacts stakeholders are interested in linking, and in how links are related.

First, stakeholders are interested in capturing links to different types of artifacts. For instance, an architect may be interested in linking design rationale and requirements to design, while a QA engineer may be interested in linking test cases to requirements. Different stakeholders may also be interested in linking to artifacts at different levels of granularity [69]. Thus, tool support for link capture must be flexible enough to enable stakeholders to link to different artifacts and to different levels of granularity. This capability can be supported with open hypermedia techniques as discussed in Section 3.4.

Secondly, variability exists in how links are related. We believe that stakeholders who are interested in capturing link are the experts on the links they capture. They are familiar with the artifacts they trace and they have a purpose for capturing the links. It follows then that the usefulness of a link is highly dependent on the stakeholder and the purpose of the link. A highly relevant link for one class of users may be irrelevant for another.

Supporting the knowledgeable capture of links has implications for the automated tool support. The tool must enable stakeholders to plug-in their own heuristics and their knowledge of how artifacts are related. This can be implemented with rules (see Section 3.5).

Insight #2: An effective traceability tool must enable stakeholders to use the links they capture. The ability to use the links depends on support for access to captured links, for analyzing captured links, and for maintaining traceability links.

First, in order for the links to be usable, it must be easy to access the captured links. Access entails rendering linked artifacts in their native editors or in a user selected editor if the artifact can be rendered in multiple editors. It may also entail rendering a specific location within the artifact such as a page, slide, or worksheet. Tool support for navigation must enable the user to specify the tools and the level of granularity to render a linked artifact (see Sections 3.4.2 and 5.2.2.3)

Secondly, support for link analysis is necessary to help stakeholders accomplish development tasks. Support for link analysis includes identifying correct or incorrect links and extracting pertinent linked information. Link analysis may also entail querying and manipulating linked information to help stakeholders identify additional tasks to complete [25]. Link analysis may be supported with a visualization wherein the link metadata is displayed. Extracted linked information may also be visually depicted on top of the architecture graph (see Section 5.2.2.5).

Third, in order for the links to be usable, links must be updateable and maintainable. Support for link maintenance enables stakeholders to identify which links have deteriorated without navigating the link and manually examining the artifact. More advanced tool support will update the link location if the artifact was moved to a different location. This can be supported with notification adapters (see Section 3.4.2 and 5.2.2.4).

3.2.3 State of the Art in Supporting Stakeholder-driven Traceability

Despite the advantages of providing tool support for controlling trace link capture and using captured links, current tools and techniques fall short of providing these capabilities. Off-the-shelf tools are generally inflexible in providing user customization [104]. Limited success has been demonstrated when a tool provides some level of extension mechanism [9]. Current traceability techniques do not provide mechanisms for user customizations [109, 70, 42]. Consequently, some organizations resort to building a custom in-house traceability tool in order to cater to specific stakeholder requirements for link capture and usage [25, 90]. While techniques exist for providing project-specific customization [48] and for providing direct benefit to users [90, 101], these approaches fall short of providing customized tool support for stakeholder-driven traceability.

Visualization of traceability links aids in analyzing captured links. ENVISION presents trace links as a hyperbolic tree to enable users to focus on one link endpoint, represented as a node, at a time while viewing all the linked artifacts in the background [125]. The tool also supports viewing transitive links, filtering and searching, recording

user navigation across the hyperbolic tree, and adding links within the visualization. TraVis enables the visualization of information from collaborative software development platforms (CSDPs) to display heterogeneous information unified by the CSDP (e.g. documents, code, Wiki pages, tracker items) [46]. TraVis extracts the information from CSDP using remote APIs or Web interfaces. Both Envision and TraVis present linked information as graphs of links that can be filtered or searched

Visualization techniques have also been used to determine the quality of candidate links. Duan uses cluster-based techniques to group trace results that are presented to the user [50]. Meanwhile, VisMatrix graphically represents the level of confidence of candidate links in a traceability matrix [49]. Other visualization techniques include tag clouds to graphically display the frequency of terms and a tree structure to represent the hierarchical structure in a requirements document [38].

While these visualization techniques are useful in analyzing captured links, these techniques do not take the extra step of extracting linked information and aggregating this information around the architecture. Presenting the links as a mashup on top of the architecture graph can provide a global view of the system and the linked information.

3.2.4. Relevance of Stakeholder-driven Traceability

In some settings, organizational interests in traceability are not aligned with and take precedence over stakeholder interests. In these contexts, project-specific customizations may be sufficient [48].

3.3 Capturing Links Prospectively

The third key element is the prospective capture of traceability links which captures links *in situ* while artifacts are generated or edited. This is complementary to retrospective techniques which recover links after the fact [17, 70]. Empirical evidence from computer-human interaction and program comprehension research communities reveal that links captured in this manner are often useful [107, 122]. In this section, we provide a brief background on prospective link capture, the rationale for using prospective link capture, the current state of the art in the automated link capture, and the relevance of the technique.

3.3.1 Background to Prospective Link Capture

The online capture of links can ensure that important information is not neglected or overlooked due to lack of resources or time [120]. The idea of prospectively capturing links is not new, but it was not until recently that prospectively capturing links has become a feasible technique [77, 97, 98].

One way to prospectively capture links is through the recording of user interaction with artifacts. Other research areas have studied the recorded user interaction to raise awareness and to support program comprehension (PC). For instance, computer human interaction and computer-supported cooperative work employ user interaction to raise awareness [73, 106, 122]. Recently, the PC community has studied the capture of user interaction to aid in program comprehension: using a team's interaction with the code to create links between source code files [47], using a developer's navigation patterns between an IDE and a browser to create links between code and documentation [63], and using navigation patterns in the code to create links between tasks and source code [79, 126]. The recording of user interaction in these different research areas suggests that this is a viable approach to capturing links between artifacts. While PC techniques have focused on user interaction with the artifacts represented as text, namely source code, links can also be created across heterogeneously represented artifacts, as we demonstrate with our exemplar implementation in Section 5.

3.3.2 Intelligent Capture or Just Noise?

This section elaborates on the rationale for the prospective capture of links.

Insight #1: Prospectively capturing links, via recording user actions, provides a vehicle for knowledgeable capture of links. The prospective approach has advantages over existing techniques. It can capture contextual information and temporal relationships between artifacts which can provide information on how the artifacts are related (see "Add relationship" rules in Section 3.5). For example, accessing a requirements document while a component is selected in a design diagram may indicate that the requirements document is related to the component.

In addition, we posit that the user interaction record is a reflection, albeit perhaps at a low level,

of the developer's understanding of how artifacts are related. A developer's navigation path reveals a developer's mental model of the system [107]. We posit that stakeholders who generate or edit artifacts also have a mental model of the relationship(s) that exist between these artifacts. Prospective link capture exploits the developer's first-hand knowledge of the artifacts in the determination of related information. This contrasts with a third party analyst tasked with a traceability activity of going through a list of candidate links to artifacts and potentially misclassifying correct links as incorrect and vice versa [71, 68]. Thus, the tool must be able to monitor user actions across the tools that are integrated into the traceability system.

Insight #2: Prospective link capture can be supported by mechanisms that minimize noise. Noise — the bane of automated capture — can be minimized through directed link capture where only a small subset of user actions will be captured. First, link capture can be directed by the user's explicit action to start recording. Secondly, during a recording session, only user's actions on the set of tools with recording adapters will be captured. Third, the recording adapter can also be implemented such that it only detects specific user actions or events, such as "open file" or "visit a hyperlink" (see Section 3.4.2). Finally, rules can be used to determine valid links before they are added to the linkbase (see Section 3.5.2).

Insight #3: Prospective link capture is integrated with software development tasks. Literature has shown that it is important to integrate traceability tasks with software development tasks in order to ensure that links are captured even during tight project deadlines [120]. Since links are created in the background across different tools, prospective link capture can be integrated with software development tasks. Prospective link capture also does not require drastic changes to existing work practices.

Insight #4: Prospectively link capture is complementary to other techniques. Prospective link capture can also be integrated with other techniques, such as retrospective and manual capture, to produce higher quality links. Since link accuracy in prospective link capture depends on user knowledge of the system, using retrospective techniques can inform the user of possible relationships with other artifacts. (See Section 5.2.2.2 for a discussion of our implementation.)

3.3.3 State of the Art in Link Capture

Retrospective capture. A prevailing technique in automatically generating links is retrospectively capturing links. Latent Semantic Indexing is a technique used to cluster related documents [84]. Data mining techniques are also used to automatically create trace links between files that are checked-in or checked-out together in a configuration management system [78]. In addition, LeanArt uses machine learning techniques to learn from users' manually created links on a small set of artifacts. This linked set of artifacts serves as a training set for LeanArt to increase the accuracy of captured links [66]. These techniques, however, fall short of capturing the actual *context* wherein the artifacts were manipulated. Our approach, besides being contextual, can be easily integrated with these retrospective techniques to arrive at potentially much higher quality links (see Integrated Search Tools in Section 5.2.2.2).

Transformations. Another set of techniques is generating trace links based on transformations or translations between artifacts. ATRIUM transforms models from requirements to architecture and generates links during the transformation [89]. Richardson and Green use a similar technique to ATRIUM in that links are generated from the program specification to the synthesized code [105]. Jouault enables user specification of trace links to be created separately from the logic of artifact transformation [77]. While these techniques also enable the capture of trace links as a side-effect to development tasks (links are captured during artifact transformation), transformation is only possible across structured or semi-structured artifacts through the use of metamodels. In contrast, our technique is not limited to tracing structured artifacts. Translators may also be used to translate heterogeneous artifacts into a homogeneous form where the links can be automatically generated [13]. Translators like InfiniTé enable the user to continue working with their current toolsets. However, not all artifacts can be translated to a common form, such as graphics and media files.

3.3.4. Relevance of Prospective Link Capture

Prospective link capture may be less applicable in contexts where failure to capture some correct

links (false negatives) has negative consequences. The richness of captured links is highly dependent on user interaction with artifacts. This drawback can be mitigated by incorporating other techniques such as search tools (see Section 5.2.2.2)

3.4 Supporting Mechanism: Open Hypermedia

This section provides a brief background to open hypermedia. We also discuss our extensions to existing open hypermedia concepts and briefly discuss the state of the art in capturing links across heterogeneous tools.

3.4.1 Concepts from Open Hypermedia

Although the World Wide Web is the most dominant example of a hypermedia system today, it is critical to separate a general understanding of hypermedia from the capabilities of the Web. The Web was constructed with some design choices that maximize scalability and extensibility (to a global scale), but limit the usefulness of hypermedia concepts. Meanwhile, open hypermedia systems (OHS) [57, 14, 15, 123, 96] provide a richer set of capabilities at the cost of some scalability and robustness. In OHS's, links are not embedded in documents and files as in the Web; rather, they are stored externally to the artifacts. Thus, OHS's can embrace a wider variety of file types and editors. Links may have more than two endpoints, in contrast to the Web's unidirectional links. Link endpoints can also be specified in flexible terms, in contrast to fixed Web links. For example, an endpoint to a piece of code may be a method name or queries over the targets that are executed when the link is traversed or examined. Additionally, links can be link targets themselves, creating meta-link structures to represent more powerful conceptual relationships. Flexible link endpoints are potentially more robust in the face of changing documents.

Consequently, open hypermedia systems offer advantages for managing and manipulating traceability links. More specifically, these advantages are the modeling of links as first class entities, the usage of an independent linkbase, and the integration of third party tools using adapters. We discuss these in the next paragraphs.

First class links with n-ary endpoints are used to represent semantically rich links. These links can store the type of relationship between artifacts, as is done in Topics Maps [4], and capturing properties about artifacts, as is done in Resource Description Frameworks (RDFs) [3]. First class n-ary links can also store the path to the tool that will render an artifact . The ability to link on links (which creates a hierarchy) enables capturing traces at different levels of granularity and abstraction.

Next, open hypermedia links are stored outside the artifacts they connect, in an independent linkbase. The external management of links enables tracing heterogeneous artifacts even though they are maintained in diverse formats with different tools. Not only does this enable stakeholders to trace artifacts without switching their tools, but it also enables tracing read-only third party artifacts. External links also enable stakeholders to define and maintain their custom trace links. The independent linkbase provides users the possibility of using a variety of techniques to explore link structures—links can be traversed in any direction, arbitrary operations can be executed on links or their endpoints, etc.

There is, however, an additional cost to maintaining links externally. Changes to artifacts require that links be updated by the traceability system. Consequently, both the traceability system and the tool adapters need to perform the extra work of monitoring changes to the artifacts in order to keep the links from becoming obsolete. While this additional work does not exist with embedded links such as those in web pages, embedded links also have a drawback of pointers that link to resources that may no longer exist (i.e. broken links).

Finally, integrating third-party tools into an open hypermedia system requires the use of an adapter. A basic integration simply requires the construction of an adapter that allows the system to identify and locate endpoints (anchors) within a target document. An adapter may use the built-in capabilities of third party tools such as keyword search or hyperlinks to locate a specific location within an artifact. More advanced integrations that allow anchor tracking and in-tool link examination and traversal is also possible, depending on the extensibility and openness of the tool.

3.4.2 Extending Open Hypermedia with Open APIs

Although the idea of linking artifacts across different tools is not new [41], it was only possible to

do so within a limited set of tools. Now, given the availability of open source software and many proprietary tools with open application public interfaces (APIs), it is feasible to automatically capture links across a broad set of off-the-shelf tools at different levels of granularity. This section describes three types of hypermedia adapters that support the capture, rendering, and maintenance of traceability links. Independent of each other, these hypermedia adapters are external to the trace tool, enabling customization of the trace links captured. The implementations of these adapters are described in Section 5.2.

Recording adapters encapsulate tool-specific recorders (discussed in detail in Section 5.2.2) that enable the prospective capture of links. Recorders minimize noise by attempting to only capture relevant user actions. In addition, recorders enable the automatic capture of tool-specific events that provide the context of how the artifact is manipulated within its native editor. Recorders may listen to events fired by a third party tool or extract the tool's captured history (e.g. a web browser's history). The events captured carry meaning that pertains to the artifact and native tool editor. These events can then be used by the rules to assign link information, such as the trace relationship (see Section 3.5).

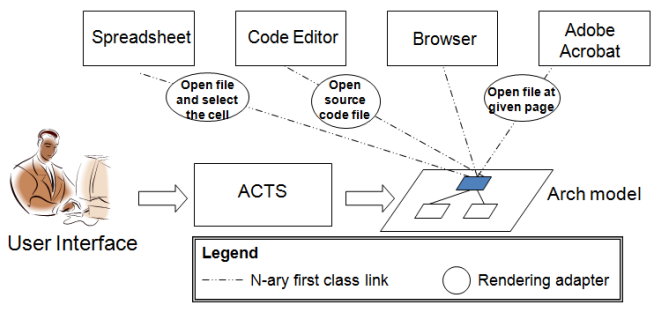

Fig 3: Linking across heterogeneous tools with hypermedia rendering adapters

Rendering adapters are used to display the selected endpoint at the specific location marked by the recording adapter (see Figure 3). For instance, rendering a cell location in a spreadsheet location entails invoking the native editor, opening the spreadsheet, and using the native editor's API to render the specific worksheet and cell location. If a rendering adapter does not exist for an artifact, the operating system's default editor will be used to render the artifact at the default location.

Notification adapters are used to monitor changes to linked artifacts in order to automatically update link metadata. For example, a notification adapter may monitor whether the bug reports linked to a component have been closed. If so, then the link status can be changed to "obsolete". Notification adapters may listen to change events when a linked artifact is opened or may be scheduled to regularly check for changes by other users.

3.4.3 State of the Art in Capturing and Rendering Links across Heterogeneous Tools

Open hypermedia based tools such as Software Concordance, InfiniTé, and Chimera enables links to be captured across tool boundaries. Software Concordance, however, requires the main representation of source code be an abstract syntax tree, instead of the programmer's native text editor, in order to effectively insert hyperlinks within source code [87]. InfiniTé is limited to tracing to artifacts that can be translated into a common text format [13]. Chimera enables users to manually capture links across tool boundaries [15]. Other tools allow the automatic capture of links across a predetermined set of tools: Codetrail (between Eclipse and Mozilla Firefox), Jazz (between Eclipse and collaboration tools), Mylar/Mylyn (between Mylar and Eclipse), and Hipikat (between Eclipse, Bugzilla, CVS repository, and a web browser) [63, 42, 2, 79]. Our extensible approach enables the integration of any third party tool as long as it provides open APIs for querying or detecting user actions.

3.5 Supporting Mechanism: Decoupled Rules

Rules allow users to determine the type of trace relationship to assign. This section discusses the usage of rules in traceability. We also discuss how rules can be customized and the state of the art in the automatic capture of custom links. (See Section 5.3 for the implementation of these rules)

3.5.1 Usage of Rules in Traceability

Rules have been used to specify link type relationship between a pre-defined set of artifacts [109, 32]. Rules, in the form of policies, may also be used to manage the link updates [88]. Oftentimes, these rules are built into the traceability system.

3.5.2 Customizing Rules

We use rules to analyze a potentially large set of events captured, to automate the capture of link information, and to support link analysis. The externally pluggable rules enable users to customize the links captured as illustrated below. The types of rules include "record" rules, "add relationship" rules, and "assign link quality" rules.

"Record" rules. Record rules analyze user interaction events to filter out noise and to generate trace links. To filter out noise, rules may be used to ignore user interactions based on an artifact type (e.g. ignore Word documents, emails, manifest files), artifact naming convention (e.g. ignore all files *ProjectX.*), or event source or event type (e.g. ignore all Save events). Rules may also be used to customize the granularity of link capture. For instance, if a user is interested in only capturing links to a spreadsheet at the cell level, then rules may be used to filter out links at the file or worksheet level.

To generate trace links, rules may use a criterion (e.g. time of access, patterns of events, primary trace artifact). Since related artifacts may be accessed concurrently or sequentially, the time that the artifact was accessed can be a basis for creating links. Analyzing patterns of events is another criterion for generating links between artifacts. For instance, patterns of interaction with a set of artifacts may indicate that the artifacts are related to each other. Finally, artifacts may be related to a primary artifact. For instance, if the architecture is determined to be the primary artifact, then artifacts subsequently accessed will be linked to selected elements in the architecture (e.g. components or connectors).

"Add relationship" rules. To add trace link relationships, rules can use contextual information such as the surrounding events captured by the recording adapters, the trace link metadata, and the surrounding software development practices. For instance, trace links captured from a web browser may be automatically assigned as "domain-specific" links. Context includes assumptions made when artifacts were generated (e.g. regulatory requirements, time restrictions) and the

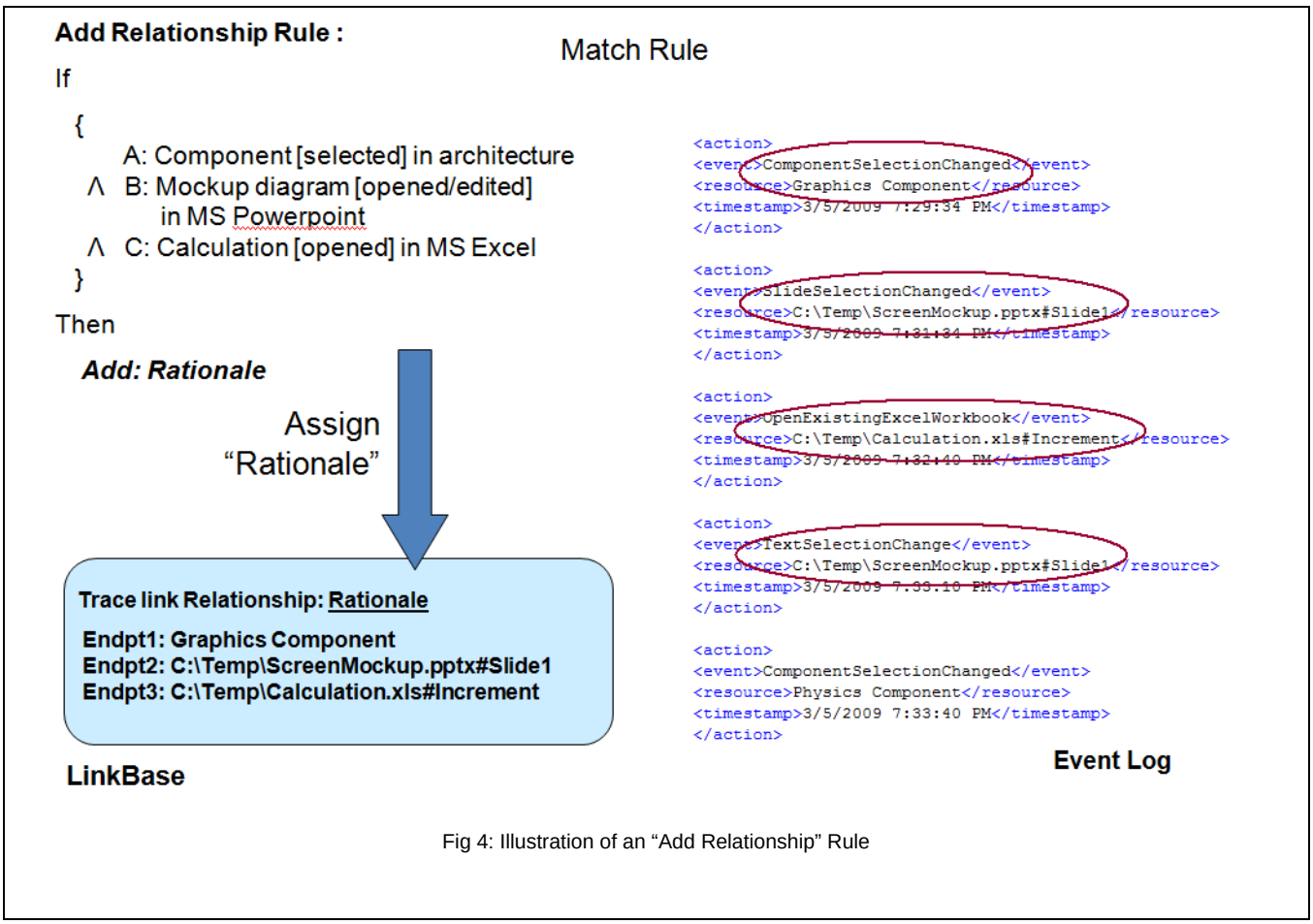

Fig 4: Illustration of an "Add Relationship" Rule

order that artifacts are generated. We posit that company conventions, procedures, or personal work habits induce observable patterns of stakeholder interaction with artifacts. In our previous work, we observed such patterns of interaction as a result of stakeholders following an established workflow [25]. These known patterns of user interaction can then be encoded as rules. When a rule matches a series of captured events, the specified link relationship is automatically assigned. In Figure 4, we show an example of an "add relationship" rule, which specifies the conditions necessary for assigning a link relationship. The trace tool processes the event log, determines whether the conditions are met, and assigns the appropriate relationship. Thus, rules can take advantage of contextual information to automatically assign trace relationships.

"Assign link quality" rules. Links may be assigned link quality based on interaction statistics and whether the links are captured using multiple methods. For instance, if a user repeatedly accesses the same set of artifacts, then the links between these artifacts would be assigned a higher quality since there is a higher likelihood that they are related. Furthermore, consider the case where links are captured by multiple methods. A link captured using both prospective techniques (using a recording adapter) and a search tool (e.g. using the Lucene search engine [18]) will be assigned higher link quality than links captured using one method.

3.5.3 Current State of the Art in Capturing Custom Links

Automated Capture of Link Types: Automatically capturing link relationships between different artifacts has been tackled by the areas of natural language processing (NLP) and information retrieval (IR). Rules can be used to automatically generate trace links with relationship types based on syntactic analysis [109]. Links are created between requirements specifications and use cases (both expressed in structured natural language) and a UML object analysis model. The rules look for patterns of terms which are assigned grammatical roles. These rules assign two types of dependency trace relations and two types of satisfiability trace relations. Camacho-Guerrero also uses NLP techniques with latent semantic indexing to automatically create semantic hyperlinks [32]. Finally, Basili et al. use co-occurrences of concepts in documents to generate typed hyperlinks [27]. Geared toward the recovery of link semantics, these approaches analyze the textual content, but not the context in which the documents are created or edited. In contrast, ACTS uses rules to examine patterns in user interaction as well as other captured contextual information. This contextual analysis enables the automatic linking of non text-based artifacts. Our rule technique also complements these text-based NLP and IR techniques.

User-Specified Heuristics: Hipikat uses various heuristics in creating links between artifacts from different sources [42]. TraCS also combines best-of-breed approaches to increase the benefit of captured links [40]. Unlike the ACTS technique, these heuristics are pre-determined.

4 Case Study: Software Acquisition

This section demonstrates the technical feasibility and utility of architecture-centric links through its application to the software acquisition domain [23]. This section briefly introduces the reader to software acquisition research and software licenses. It then shows that license links to the architecture, albeit a simplistic link in the form of an annotation is necessary to support automated license conflict analysis.

4.1 Background in Software Acquisition Research

Software acquisition research is concerned with increasing the quality and reliability of software-intensive systems obtained from subcontractors or various off-the-shelf components [11]. There is a growing trend of composing software systems using third party components with different licenses to lower development costs. These components may be open source software or proprietary tools with open APIs [117]. This strategy, however, may result in substantially higher liabilities from incompatible licenses. Consequently, the ability to identify the origin of source code, ascertain its license, and analyze license interactions within a system is necessary to mitigate liability costs. The resulting system may not have a license that resembles an existing license type [12]. Unity is an example of a heterogeneously-licensed system [117].

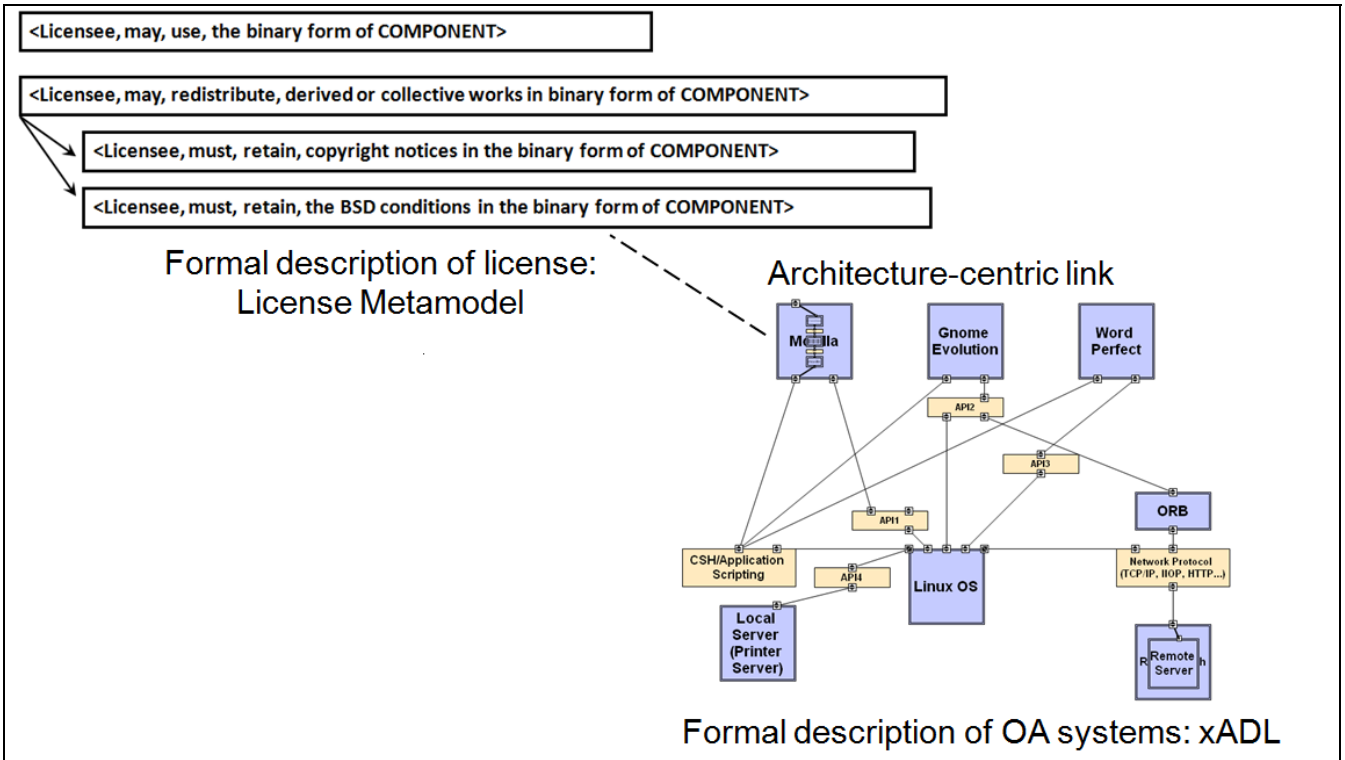

Fig 5: Architecture-centric links facilitate automating software license analysis

4.2 Background in Software Licenses

There are numerous license types, variants, and versions [93]. Consequently, analyzing the compatibility or lack thereof between the various licenses in a system is extremely difficult. License types include General Public License (GPL), Mozilla Public License (MPL), Apache Public License (APL), academic licenses such as Berkeley Software Distribution (BSD) and MIT, Creative Commons, Artistic, and Public Domain. Each license type can have multiple variants and these variants can evolve over time, resulting in new versions over time.

In addition, manually analyzing licenses is difficult because of the way they are expressed. Licenses are often incomplete and legally ambiguous or exact but difficult for individuals without a legal background to comprehend.

Furthermore, the license of the overall system may be affected by component configuration or software maintenance [12]. Components that are dynamically linked at runtime may not be included in the software release; thus, their licenses need not be included in the overall system license. Software maintenance, such as using alternative components with different licenses, changes the overall system license. In addition, using different connectors, such as replacing a procedure call with an HTTP request, can alter the overall system license.

It is in this context that architecture-centric traceability is shown to be particularly suited. The linking of license information, formally expressed as a license metamodel, to the system information, as represented by the structural architecture or xADL, enables system designers to understand the design and license tradeoffs to allow for the system's redistribution and licensing (see Figure 5).

4.3 Automated Software License Analysis

The architecture is the central artifact that enables the analysis of whether license constraints, such as legal obligations, are satisfied. Automated analysis of system properties such as adherence to communication constraints is currently supported in ArchStudio [45]. This type of analysis is solely based on information encapsulated by the archi-

tecture. Adding information to the architecture through links to external information enables a wider range of analysis on the system. In this case, linking license information to the architecture elements, specifically components, facilitates the automated license analysis. For instance, it is possible to calculate the scope of a reciprocal obligation imposed by a component with a GPL license. This calculation is simply done by traversing the architectural graph and including all the connected components that are not separated by a license firewall [11]. It is also possible to calculate system-wide obligation conflicts by traversing the architectural graph. For each visited component, one should traverse the link to the license information in order to extract the license obligation. The obtained union of all the license obligations can then be analyzed for conflicts. Similarly, it is possible to find the overall system rights by taking an intersection of all the rights in the system. Many other types of analysis are made possible by linking the architecture and software license information. For a more detailed discussion of these heuristics, the reader is referred to [12].

To demonstrate the possibility of automating software license analysis, the ACTS Traceability System has been extended with a Software Architecture License Traceability Analysis module (see Figure 6). This allows for the specification of licenses as a list of attributes (license tuples) using a form-based user interface in ArchStudio4.

The tool has been used to analyze a heterogeneously composed system that is characteristic of a typical e-business system. The system depicted in Figure 6 has three different licenses: GPL, BSD, and Corel Transaction License (CTL). Running the license analysis produces the report shown at the top of the figure. The tool is able to support linking licenses at different levels of granularity: at the component level and at the subsystem level. It can also support analyzing license interaction across these different levels of granularity. In this figure, for example, the GPL license propagates to all the subcomponents of the Mozilla component: GUIDisplayManager, GUIScriptInterpreter, and mozilla.

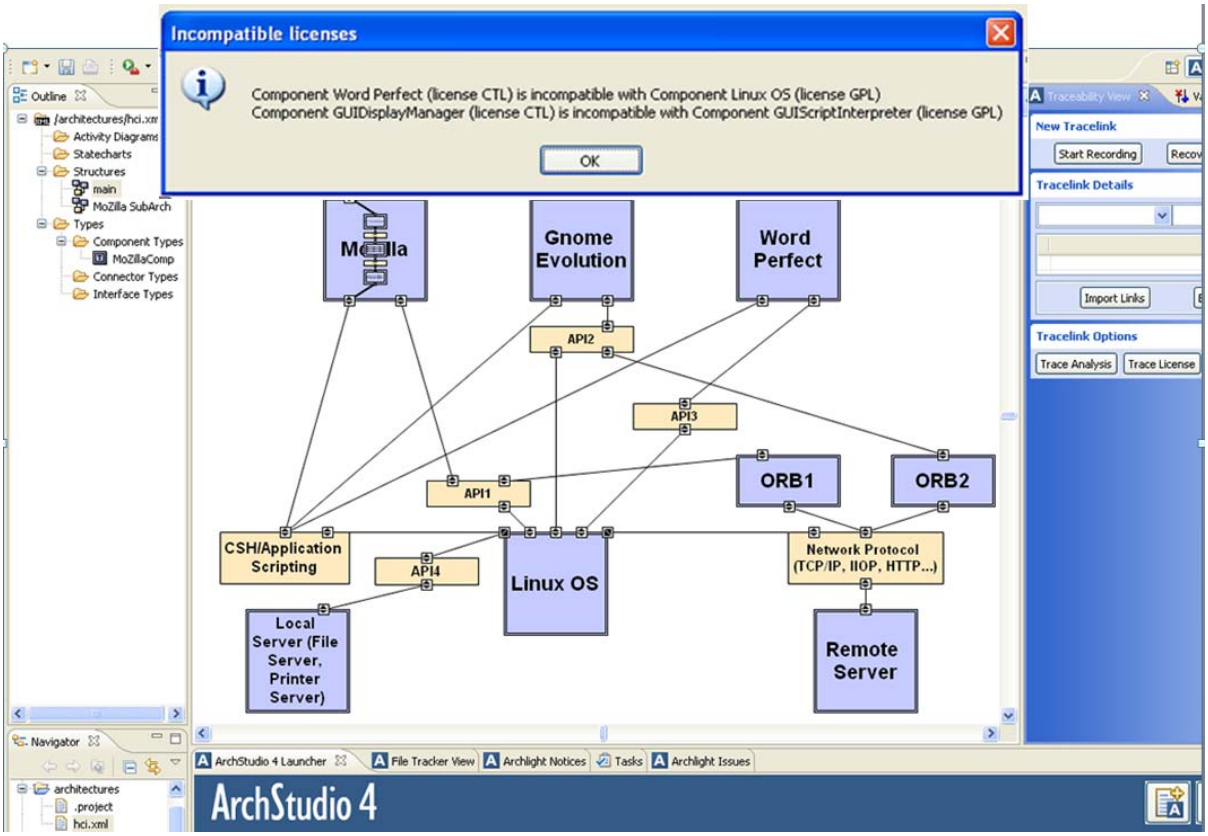

Fig 6: Software Architecture License Traceability Analysis module

4.4 Discussion

This section discussed a specialized application of architecture-centric traceability in the context of software acquisition research. This section has illustrated that architecture-centered traceability links not only provide a system view of traced information, but they can also be used to support automated analysis of system properties, which in this case is the license compatibility of heterogeneous components in a system. In this context, architecture-centered links coupled with a formal description of the software system (i.e. xADL) and a formal description of software licenses provide a solution to an otherwise intractable problem. Automated analysis is necessary in the face of evolving software licenses and changing organizational policies regarding acceptable software licenses.

Other approaches have focused on analyzing software licenses or on reverse engineering [61, 118, 116], but have lacked the capabilities of providing tool support for automatically analyzing license interactions within a system, especially during design time.

Beyond the software acquisition domain, architecture-centric traceability links can also support the analysis of other system properties. Example analyses are determining the level of coupling between components in an architecture (links between architecture and source code), the level of "bugginess" of a system (links between architecture and bug tracking repository), the level of dependency on third-party software (links between architecture and source code), and test coverage of a system (links between the architecture and quality assurance or QA test reports). The augmented trace information can also support semi-automated analysis of correctness with respect to requirements. For example, a functional requirement may state "Environmental sensors must perform time synchronization at regular time intervals". A baseline analysis can be performed by traversing the architectural graph. For each environmental sensor component, the component mapping to source code can be used to locate a "timeSynchronization" method. The lack of such a method can be a baseline indicator that the requirement has not been satisfied. If the method exists, then architects can then proceed to determine the correctness of the method. Thus, architecture-centric traceability, while only scoping the capture of links to the con-

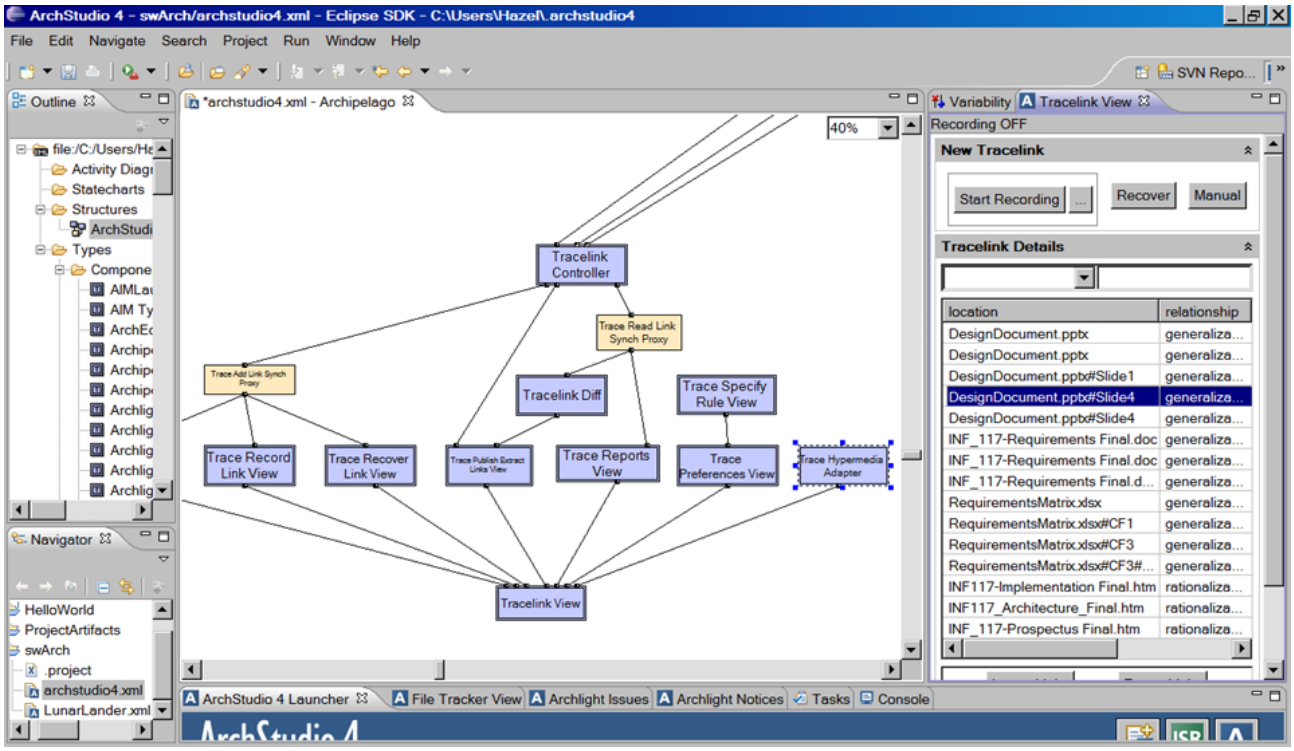

Fig 7: ACTS View displays the traceability links when a component is selected. ACTS is built on top of ArchStudio4.

cepts presented in the architecture, is technically feasible and has utility.

5 An Exemplar Implementation

While the last section demonstrated the technical feasibility of architecture-centric links, this section shows the technical feasibility of supporting the capture of user-customized links and the prospective capture of links across heterogeneous artifacts. The first subsection provides an overview of prospective capture and the overarching design goals followed. Details of the design and implementation of the tool's supporting mechanisms (open hypermedia and rules) follow in the next two subsections.

The tool development focused on capturing links to and from the structural representation of the architecture. The ACTS Traceability System is built on top of ArchStudio [45], an architecture-centric development environment that is integrated with Eclipse. Users can access the traceability support through the ACTS View in ArchStudio (see Figure 7).

5.1 Overview and Design Goals

5.1.1 Overview of Prospective Link Capture

Prospective link capture is supported by open hypermedia recording adapters and rules. Figure 8 shows an overview of the process of prospectively capturing links -- the numbers denote steps, triangled steps denote user actions, and circled steps denote automated tool support. A user may select which rules to apply prior to any recording session (Step A). The user initiates a recording session in the trace tool (Step 1). The trace tool invokes appropriate tool-specific recorders (Step 2) whenever the user opens specific artifacts. As the user performs development tasks and accesses, generates, or edits artifacts (Step 3), each recorder captures the user interaction events. Each event captured is associated with the resource path and optionally a location within the resource. When the user ends the recording session of the trace tool (Step 4), the adapters output the captured events to a common event log (Step 5). The trace tool orders the events sequentially. Rules may be automatically applied to transform the event log into traceability links (Step 6). Finally, the new traceability links are added to the linkbase (Step 7). Users are not required to validate the links as a separate task. As they go back and use the links, users are allowed to remove any invalid links they encounter.

The level of granularity of link capture is dependent on both the artifact and the tool's APIs. For instance, Eclipse allows recording at both the file level, such as a file selection in the Navigator View, and at the element level, such as an element selection in an editor view. Meanwhile, MS Excel allows recording at the file level, at the worksheet level, and at the cell level. Since recorders are external to the trace tool, the granularity of recording may be user-customized.

5.1.2 Lightweight, Customizable, and Integrated Link Capture

We built the open hypermedia adapters and rules with the goal of supporting lightweight, customizable, and integrated link capture.

Lightweight. The tool is designed to limit the overhead in the tool setup and the link capture. In contrast to previous prospective approaches where a development process needs to be specified [98] or where all possible links between artifacts are pre-specified [97], the setup is limited to the tools and heuristics the user is interested in integrating into the trace environment. A tool adapter must also be constructed for each third party tool to be integrated into the ACTS Traceability system. We minimize overhead in the tool usage through the background capture of links while users perform their development tasks.

Customizable. The tool is also designed to be customizable. It supports the user-directed capture of links through selective hypermedia recording of user interaction, and the selective generation of links from the recorded user interaction. Link information is also automatically assigned via rules (detailed in Section 5.3). Furthermore, users may choose the level of interaction with the tool. They may have the record button turned on all the time or explicitly switch to the record mode whenever they choose to capture links in the background.

Integrated. Unique to the ACTS tool is the externalization of the recording mechanisms and the heuristics used to generate trace links. The external hypermedia adapters make it possible for third party tools to be integrated into the trace tool. Moreover, external heuristics in the form of rules also enable users to integrate their custom heuristics. Once users specify the location of their cus-

tom rules and adapters, they are integrated into the system.

5.2 Supporting the Capture and Usage of Links

Open hypermedia techniques and mashups are used to support the capture, usage, and analysis links. Hypermedia adapters capture and render links within their native editors. Link usage and analysis is enhanced by mashups which graphically renders the extracted information from the linked artifacts.

5.2.1 Increasing Openness, Accessibility, and Usability

The tool is designed to support the capture of links across distributed information residing in heterogeneous tools. It is important to note that not all user interaction events are captured, but only those that can be intercepted by the hypermedia adapters. This section discusses how the ACTS approach achieves openness and enhances accessibility and usability of linked information.

To increase openness among heterogeneous tools, the tool has explicit extension points where users can integrate their tool-specific hypermedia adapters. Users simply add the path to their custom adapter to the trace tool's list of hypermedia adapters. When the recording session starts or when the links are traversed, the tool automatically invokes the appropriate adapter.

To increase accessibility, we implemented the following. All the trace links are presented through a unified interface, which is the graphical rendering of the architecture. Links to artifacts at different levels of granularity are supported (e.g. file level, page level, section level) to facilitate accessibility to specific locations within an artifact. Linked artifacts are also rendered within their native editors.

To increase the usability of captured links and to facilitate link analysis, mashups are designed to overlay linked information to the architecture. It not only supports the traversal of the captured links, but it also extracts the information from the captured links to provide users a comprehensive view of the system along with the related information.

5.2.2 Tool Implementation

This section discusses the implementation of first-class n-ary traceability links, and hypermedia recording, rendering, and notification adapters. Mashups are also used to provide link visualiza-

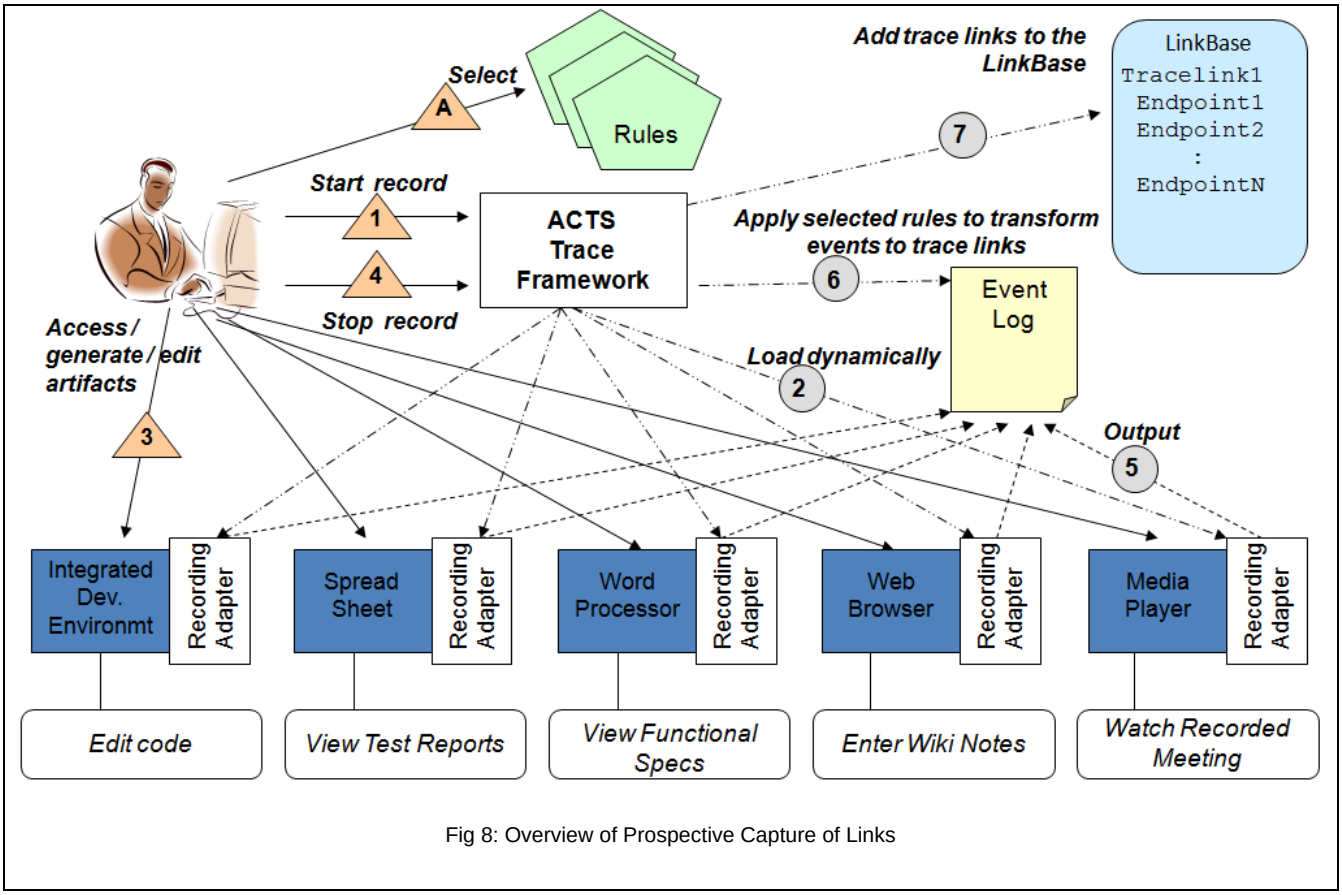

Fig 8: Overview of Prospective Capture of Links

tion.

5.2.2.1 First Class N-ary links

First class n-ary links group artifacts with a common relationship (e.g. satisfaction, rationale). These first class links not only store link information, but they also support modeling link hierarchies to present linkages between course-grained and fine-grained artifacts. A trace link consists of a set of two or more endpoints. A trace endpoint includes an artifact location, timestamp of link capture, method of link capture, and it may optionally include link quality status, the user who captured the link, the custom action performed when navigating a link, and a link to another trace link object. Custom actions point to a script or executable code that is launched when the link is navigated.

The traceability links are stored in a xADL file, an XML-based architecture description language [44]. xADL was extended with a traceability schema extension.

5.2.2.2 Recording Adapters and Uniform Event Model

Recording adapters were implemented for Eclipse 3.4, Microsoft Office 2007, Adobe Acrobat 9, and Firefox 3.

Eclipse 3.4: The Eclipse recorder listens to SelectionEvents fired when user selects elements within the Eclipse Views and Editors. When these events are fired, the recorder obtains an ISelectionModel which specifies the selected element and the view or editor where the event was fired.

Microsoft Office 2007: Microsoft Office tools fire events when the user modifies the artifact, selects a section of the artifact (e.g. a slide or a page) or invokes built-in commands (e.g. open, close, save). The recording adapters for Microsoft Office listen to these events to detect user actions. We built a different adapter for the following tools: MS Word, MS Excel, and MS PowerPoint. Each adapter is implemented as a standalone Visual Basic executable, although they could have also been implemented in C++ or C#.

Adobe Acrobat 9: Adobe Acrobat does not provide an API for listening to user navigation. Instead it provides APIs, called the Interapplication Communication (IAC) [6], for enabling third party developers to programmatically invoke the capabilities within Adobe Acrobat. The recording adapter for Adobe Acrobat simply extracts all the user changes to the Acrobat file along with the change timestamp. The adapter is implemented as a Visual Basic executable.

Firefox 3: Similar to the previously discussed tools, Mozilla Firefox provides mechanisms for customizing the capabilities of the browser. We implemented a recording adapter that captures course-grained links to visited sites by directly querying the browser's history database, stored in a SQLite database [5]. The recorder extracts the visited sites along with their timestamp. We also implemented a prototype Firefox adapter which uses Mozilla's XML User Interface Language (XUL) and JavaScript [86]. This adapter extends the browser's capability to listen to finer-grained user actions within a webpage such as button clicks and mouse-over text actions. This adapter is also more flexible since it is not affected by changes in the underlying data model of the tool.

Uniform Event Model: Once the recording session is completed, a recording adapter stores the recorded events into an XML event log file. In order to unify the extracted information from the different recorders, a standard data model is used. An action tag represents a recorded user interaction. Each action is associated with the detected event, the resource, and the timestamp. The resource represents the path of the artifact on which the event was detected. The "#" sign delimits the path to the artifact and the specific location within the artifact. The selected item in Figure 7 shows the file path and slide number in the PowerPoint file.

Integrated Search Tools: We integrated Lucene, Trac and Google into our ACTS framework. Lucene is a well-known third party tool that provides text matching between sets of documents [18]. When a component or connector is selected, we use Lucene to link to all the documents in a given file directory with the matching component or connector name. We also integrated the Trac issue tracking system [51]. When a component or connector is selected and a link to the Trac repository is navigated, a query is automatically invoked to display the issues reported against a selected component. Finally, we also integrated Google search for searching artifacts on the Internet.

Google may also be used to guide the prospective capture of links. An example scenario is (a)

using Google to find related links, (b) turning prospective capture on, and (c) recording which candidate links generated by Google are actually used – perhaps repeatedly – by the architect.

We have also illustrated with our more recent work how prospective capture can be combined with an advanced machine learning technique called topic modeling [24].

Implementation challenges of recording adapters: Implementation challenges include supporting extensibility, intercepting user interaction, building platform and version independent adapters, and synchronizing time.

The first challenge is supporting extensibility to accommodate various third party tools. This challenge is addressed by using a combination of a procedure call and shared repository. When users open a file, the ACTS environment calls and passes control to the appropriate recording adapter. While the recorder is running, interaction events are stored in a shared repository. When the user closes the application, the recorder shuts down and hands control back to the ACTS tool. The ACTS tool then takes the interaction events from the shared repository and transforms them into links. Thus, the integration of third party tools is greatly simplified by the indirect data transfer from the external recorders to the tool.

The second challenge is in regards to implementing the recording adapters to intercept the user interaction with the third party tools. Some of the integrated third party tools do not have public APIs for listening to user navigation within the tool. In implementing the Adobe Acrobat adapter, only specific changes to the file are intercepted, such as comments or strikethroughs. Consequently, links to specific locations within Acrobat file may only be captured if the file is modified. Meanwhile, in implementing adapters for the Microsoft Office suite, it is necessary to get a handler to the specific file that the user is editing in order to be able to listen to the commands invoked by the user.

Still another challenge is building platform and version independent adapters. Recording adapters depend on the third party tool's API or data model and are thus sensitive to the tool's changes. Such was the case with Adobe Acrobat and Firefox. The adapter for Adobe Acrobat 9 will not work with earlier versions because it is using advanced features that are specific for version 9. Similarly, Firefox Mozilla's data model changed between Firefox 2 and Firefox 3. Thus, it was necessary to create a new adapter for Firefox 3. A more elegant adapter for Firefox is to use XUL and JavaScript that would be independent of the internal Firefox data model. This adapter, however, is still subject to changes in the Firefox API. The adapters for some tools can also constrain the tool to a specific platform. For example, hypermedia adapters we built for MS Office require that the ACTS trace tool run on the Windows Operating System because OS libraries are used to obtain a handler to the file being accessed. A different set of adapters is then needed to have the ACTS trace tool and MS Office adapters run in a different operating system. Thus, adapters are currently limited to the version or platform for which it is developed.

Finally, time synchronization is an important issue when integrating the various events from the different recorders. It was observed that the different recorders were synchronized, since they were running on the same host, but they store the time in different formats. Consequently the Java Date class which represents the Universal Time (UT) was used as a standard time. The various recorders were modified to translate their default time into the Java time format. If the recorders were running on different hosts, then it is important to account for any time differences between the hosts.

5.2.2.3 Rendering Adapters

This section discusses the corresponding rendering adapters for the tools discussed in the previous section. To render an artifact, the ACTS Traceability System first checks the path of the link to be traversed and then invokes the appropriate rendering adapters. Analogous to the anchor concept within a webpage, the adapters render the artifacts to a specific location within the document if a "#" delimiter exists in the linked artifact.

Implementation challenges of rendering adapters: Implementation challenges, such as minimizing lag and identifying an anchor within the artifact, were encountered. Course-grained rendering support (at the file level) was initially provided for Word documents, Excel spreadsheets and PowerPoint slides within Eclipse workbench default editor. However, due to the observed per-

formance lag and difficulties with manipulating the artifact within Eclipse, the rendering adapters were implemented outside of Eclipse.

Another challenge is in the fine-grained rendering of artifacts. The rendering capability is limited by the third party API. For instance, the MS PowerPoint renderer is limited to rendering at the file and slide level while the Excel renderer can render at the file at finer levels of granularity: worksheet, row, column, and cell levels.

5.2.2.4 Notification Adapter

To support link maintenance, we implemented a notification adapter for MS Excel 2007. Implementing a notification adapter requires determining how to handle the different types of updates as well as determining the frequency of updates.

We define three types of updates: deletion, revision, and relocation. Deletion is when the resource has been deleted within a given search space. Revision is when the resource has been modified. Relocation is when the resource has moved to another location. These updates may take on different forms, depending on the level of granularity of the linked resource. (See the Implementation Challenges below for details on how these updates are handled.)

We also determined the frequency of updates for our notification adapter. Updates can be performed as a result of continuous monitoring of the linked artifacts or on demand update. We opted for the on demand update since it is a more efficient approach in terms of processing time required. When the user is ready to use the links, the user can simply invoke the notification update to determine which links have changed since the last session.

Implementation challenges of notification adapter: We discuss issues encountered in our implementation of the deletion and relocation updates. Although we did not implement the revision update, we offer some implementation insights.

The notification adapter handles updates at the same level of granularity as the link captured. Deletion updates are handled at the file level, worksheet level, and cell level while relocation updates are handled only at the content level. The current implementation handles updates at these levels differently. If the link is at the file or worksheet level, the notification adapter checks if the file or the worksheet exists at the specified path. If not, the link status is updated as “deleted”. If it does exist, the link status is left blank, indicating that it is a valid link. If the link is at the cell level with no specified content, then the adapter checks if the specified worksheet exists. If the link is at the content level, then the adapter checks if the content is found at the specified cell. If it is found, then the link is valid and the status is left blank. If it is not found, the adapter searches for the content throughout the file. If a match is found, the status is updated to “moved” and the link itself is modified to point to the new cell location. If no match is found the status is updated to “deleted”.

Implementing relocation update is a function of the search space. In the case of our Excel notification adapter, handling the relocation case at the content level was fairly straightforward. It simply required searching for the given content within the Excel file. To implement the relocation update at the file level, one must decide if the search space is within a given directory, within a given machine, within the company intranet, or within the Internet. As the search space increases, more sophisticated searching algorithms become necessary.

Implementing the revision update is a function of the user interest in the change and the granularity of the resource. If a change has been made to the text font or color of the resource and the user is not interested in tracking this change, then from the user’s perspective, the resource has not changed. Otherwise, the adapter must detect the change and update the status to “revised”.

The granularity of the resource is another factor in handling the revision update. The courser grained the linked artifact, the more sophisticated the change detector must be. To implement a revision update at the cell level, it would simply check if the value of the cell matched the content that it was intended to point. This could simply be a string or a numerical match test, depending on the cell content. However, in the event that the resource is at the file level, a different technique is needed to determine if the file has changed. One could do a simplistic comparison of the file modified date, a more sophisticated check of the latest changes made based on a repository check-in, or a still more sophisticated scheme of running a diff

algorithm on a snapshot of the previous linked artifact and the current artifact. The last option would require more storage overhead for storing the artifact's snapshot.

5.2.2.5 Aggregate Link Information

Once links are captured, the linked information can be aggregated and presented to a wider range of users visually as a mashup to support link usage and link analysis. A Mashup Link Processor and an Information Extractor which queries the Trac Issue and Bug Tracking System were implemented. The Mashup is implemented as a server-side Flash CS4 script. The user simply loads the xADL file which contains the architecture as well as the linkbase. The Flash script then renders the architecture in a browser. Figure 9 shows the mashup of the ArchStudio 4 architecture with bug and source code information overlaid on top of it. Blue shading represents components and yellow shading represents connectors. The border shading represents whether the components have reported bugs (yellow), lack source code (red), or have source code and no reported bugs (black). The figure at the top shows an actual bug reported against one of the ArchStudio components, Launcher. When a user clicks on a component, a pop-up table of the traceability links appears. A user can then navigate to a linked artifact. In this example, a link to a reported bug displays the bug within the Trac bug database. This example illustrates how linked information can provide real-time project status as far as component implementation and component errors are concerned.

Implementation challenges of aggregating link information: A couple of performance issues were encountered in the implementation of the mashups: 1) parsing the xADL which contains the linkbase and 2) data extraction especially from remote servers. To address the first challenge, the mashup stores the parsed xADL file so that the repeated rendering of the unmodified file does not require reparsing the xADL file again. To address the second challenge, a timeout was used to limit the user wait time and to only render the available information prior to the timeout. In addition, HTTP requests to SVN and Trac bug tracking system were placed on parallel threads to minimize the lag in extracting the linked information.

5.3 Using Rules

Rules encapsulate the heuristics for creating links, filtering links, and capturing link information. This section covers the implementation of rules.

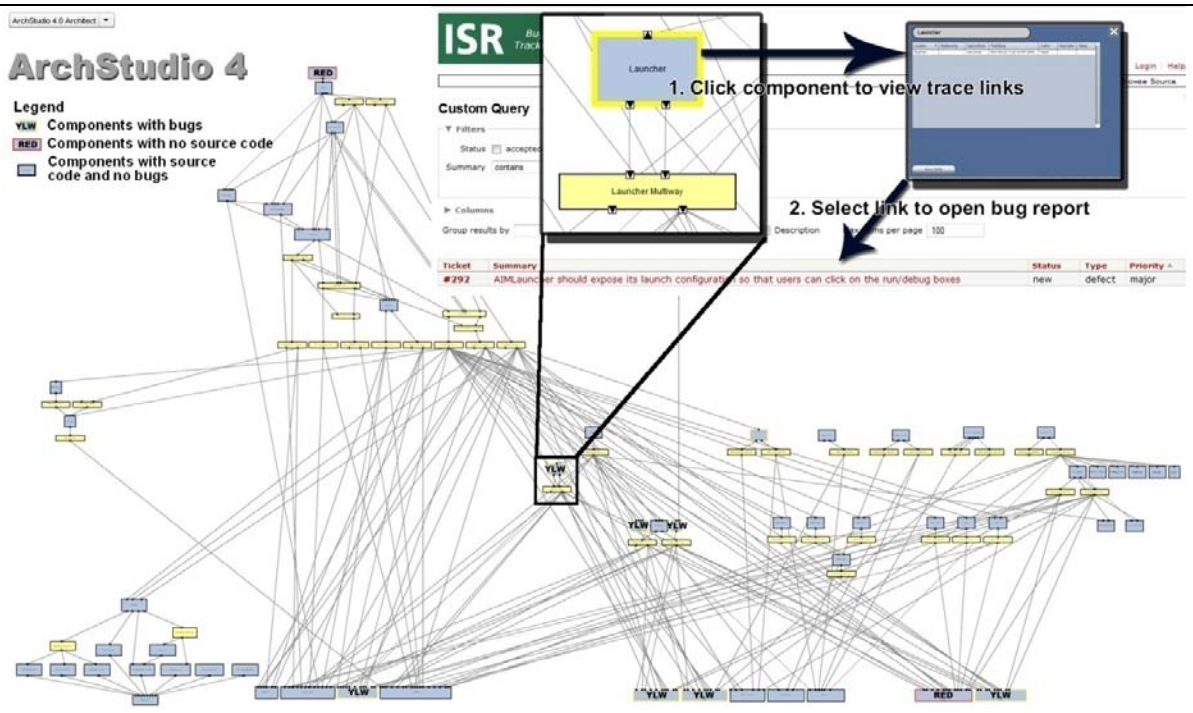

Fig 9: Sample Mashup of ArchStudio 4

5.3.1 Adaptable, Modular, & Contextually-Aware Rules

The tool is designed to support the custom capture of trace links through the use of adaptable, modular, and contextually-aware but content-blind rules.

Adaptable rules: In order to support the custom capture of traceability links, users may adapt the rules according to their heuristics. Rules, which are external to the tool, may be created for a class of users or for individual users. Users simply specify in the tool which rules to apply when capturing traceability links.

Modular rules: A rule is a self-contained heuristic unit. This modularity enables users to easily evolve each rule over time. This modularity also enhances the interchangeability of rules, enabling users to swap rules when they change their heuristics. Rules may also be combined with each other to form more complex rules.

Contextually-aware but content-blind rules: Designed to be contextually-aware but content-blind, rules operate on the captured contextual information, making it possible to create links and assign link types to heterogeneously represented artifacts. Rules are content-blind since the basis of link creation is not the content of the artifacts, but the users' activity surrounding the creation or modification of artifacts. Rules, working in tandem with open hypermedia adapters, enable custom links to be captured across heterogeneous artifacts. This is an important distinction from previous traceability approaches that tended to employ content-aware but context-blind approaches. Techniques from natural language processing (NLP) and information retrieval (IR) are excellent in determining possible links using the textual content of an artifact [109, 32, 27]. Meanwhile, contextual awareness enables existing development processes, company conventions, or developer work habits to be loosely integrated into the link capture without heavily or completely specifying a development process.

5.3.2 Tool Implementation

Rules are used to create traceability links and to capture the trace link information. Current rule support includes "record" rules (filter events rule, generate trace links rule) and "add relationship" rules. The rules are currently implemented as XSL Transformations (XSLT) on XML; Xalan-Java acts as the rule engine [19]. (Alternatively, an off-the-shelf inference engine could be used.) XSLT has been commonly used to transform XML documents. In the ACTS Traceability System, XSLT is used to encapsulate the action to take (i.e. transformation on the event log file) when a pattern of events has been detected. Usually, a rule is represented by a single XSLT file, while some rules may span multiple XSLT files.

Users also have the option of either interactively applying the rules or applying them in the background. For background rule application, users can specify the rules to apply prior to any recording session (as shown in Figure 8). For interactive rule application, users select the rules interactively after each recording session. A dialog box shows the status of the rule application and prompts the user for additional rules to apply.

When the recording session is completed, the captured user interaction logs of the hypermedia recording adapters are ordered by time to recreate the sequence of user interaction across different tools. Rules are then applied to transform the event logs into trace links. Rules that filter events may be first applied to the event logs to eliminate the noise captured. Furthermore, rules may be applied after trace link generation to remove any duplicate links to the same artifact. The duplication of links occurs when multiple commands are invoked on the same artifact.

"Record Rule": We can generate trace links based on grouping by architectural elements. After a component selection, all the artifacts that the user selected is automatically linked to the selected component in Archipelago, ArchStudio's graphical editor. The rule checks if the selection is an architectural element in the Archipelago Editor. If so, the assigned group number is incremented by one to indicate that the architectural element and the succeeding artifacts are assigned a new group number. Otherwise, the group number remains unchanged.

"Add Relationship" Rule: Figure 10 shows an excerpt of the rule that assigns the type of trace link relationship based on the file format of the artifacts (see lines 174 to 192). A more complex rule is illustrated in Figure 11. This rule assigns the trace relationship type based on the set of surrounding artifacts that the user accessed. In this particular example, if the set of accessed artifacts

includes a PowerPoint file and a graphic file (i.e. jpg), then the rule assigns a rationalization link type. If the set of accessed artifacts only include a PowerPoint file, but not a graphic file, then the rule assigns the generalization link type. The link types are currently arbitrarily assigned by the rule author. Future implementations may incorporate alink type ontology or a classification scheme.

Implementation Challenges of Rules: Several implementation challenges were encountered in transforming captured events to links and filtering the captured links. Possible scalability issues and understandability issues in writing rules are also discussed.

Transforming Events to N-ary Links: It is challenging to group selection events to create a valid set of trace links. Since we use a more flexible means of grouping artifacts, n-ary linking as opposed to bidirectional links, it is necessary to determine the boundary of a link. Selections may be grouped by time, by primary artifact, or by explicitly turning on or off of the record mode.

Grouping events by time entails grouping artifacts based on the time of the previous event to time of the current event. If the time difference is within a certain period, the artifacts are linked together. This proved to be an inflexible approach since users can easily exceed the time group boundaries.

Grouping events by primary artifacts (e.g., using architecture elements like components and connectors as group boundaries) is a reasonable approach, since artifacts may be related to a primary artifact. This approach is not necessarily restricted to the architecture as a primary artifact, but may be applied to any primary artifact as long as the elements within the artifact are uniquely identified.

Grouping events by explicitly by turning the record mode on and off is an approach that collects all the artifacts captured during the recording session as a set of endpoints belonging to one (large) trace link. This means of grouping events requires manual intervention, but it addresses the difficulty with grouping different units of a primary artifact together. Thus, any number of elements may be linked to each other as well as other artifacts.

Grouping events using navigational cycles is another means of determining links between artifacts [107]. Accessing a previously visited artifact creates links between the artifact and all other artifacts visited prior to the return visit.

Filtering Events: It has been observed that multiple filtering mechanisms are needed to effectively remove noise and ensure that correct links are not discarded. One example is filtering by timestamps. This filter can remove jitters, quick movements across artifacts [107], or unnecessary events captured by a tool like Eclipse. For each discrete Eclipse event detected, prior selections in a view are captured as well as new selections. In order to filter out the prior selections, a timestamp filter is used. However, filtering by time can po-

```
170
171     <xsl:otherwise>
172      <!-- reached the top of the group -->
173      <xsl:choose>
174        <xsl:when test="contains($curFileType, 'htm') or contains
175          <xsl:value-of select="string('rationalization')"/>
176        </xsl:when>
177
178        <xsl:when test="contains($curFileType, 'ppt')">
179          <xsl:value-of select="string('generalization')"/>
180        </xsl:when>
181
182        <xsl:when test="contains($curFileType, 'Chapter')">
183          <xsl:value-of select="string('domain')"/>
184        </xsl:when>
185
186        <xsl:when test="contains($curFileType, 'wiki')">
187          <xsl:value-of select="string('implementation')"/>
188        </xsl:when>
189
190        <xsl:otherwise>
191          <xsl:value-of select="string('')"/>
192        </xsl:otherwise>
193
194      </xsl:choose>
195
196    </xsl:otherwise>
```

Fig 10: Add Relationship Rule

```
167
168     <xsl:otherwise>
169      <!-- reached the top of the group -->
170      <xsl:choose>
171        <xsl:when test="contains($curFileType, 'ppt')">
172          <xsl:choose>
173            <xsl:when test="contains($curFileType, 'jpg')">
174              <xsl:value-of select="string('rationalization')"/>
175            </xsl:when>
176            <xsl:otherwise>
177              <xsl:value-of select="string('generalization')"/>
178            </xsl:otherwise>
179          </xsl:choose>
180        </xsl:when>
181
```

Fig 11: Add Relationship Rule based on co-accessed artifacts

tentially eliminate valid links. This issue can be addressed by coupling a timestamp filter with other types of filters (e.g. filters that check for the selected elements), or using event patterns as a basis for filtering selections.

Scalability: The current rule implementation, which uses the XSLT engine, may incur performance overhead in processing thousands of events. To mitigate this performance overhead, offline processing of events and rule engines can be used.

Understanding Rules: Rules are currently expressed as an XSLT which is based on template matching as opposed to the more commonly used procedural programming paradigm. Rules, which require examining the elements before or after the current node, may require the use of recursion to properly transform the event log file. Thus, writing rules may not be accessible to some users. This difficulty can be addressed by having a technically proficient trace tool administrator who can write the rules for the other members of the development team. Another possibility is to develop a user interface that allows users to graphically specify the rule and have the tool automatically generate an XSLT files. Off-the-shelf tools that support the editing and debugging XSLT files are also available [113, 102].

6 User Feedback

To help evaluate the usability of our approach, we solicited the feedback of 33 users: 12 from industry, 18 PhD students and 3 undergraduate students in either Computer Science or Informatics at the University of California, Irvine. Feedback was obtained either through online surveys, paper surveys with unstructured interviews, or personal communication. While this study is limited and artificial in several respects, it did yield worthwhile feedback.

Our study did not fully investigate the ability of users to control link capture through custom rules or the ability of the tool to adapt to different settings by integrating user selected tools. The tool was used in various settings (personal projects and trial usage) with a set of readymade rules. Some users only used the Firefox adapter to link to online resources, while others used the MS Word and Adobe Acrobat adapters to link to local resources. One participant used the MS Excel adapter.

The users were asked to perform the following tasks. The users were asked to use ArchStudio and the ACTS View to capture links while they edited a structural design and viewed or edited documentation files. Some of the users were asked to manually apply four rules while others had the rules applied in the background. All users were also asked to retrieve the captured links.

We sought to understand user perception on usefulness of architecture-centric links, overhead of capturing links, ease of accessing artifacts, tension between automated and user-controlled capture, privacy concerns, and tool usability.

6.1 Usefulness of Architecture-centric Links

Linking artifacts to the architecture (or design) is a useful feature to some of our participants. Eight participants (three of whom have industry experience) expressly identified this feature as the feature they liked about the tool. Two participants with industry experience, however, said that linking to the architecture was not applicable to their work contexts. Still another industry participant said that the tool has “potential use in Safety Critical Applications”.

6.2 Overhead in Capturing Links

The participants generally felt that the time spent in capturing links was acceptable. One user commented that the tool “make[s] the linking job easier” and another user stated “The tool saved me lots of time. Thanks!”

Users who applied the rules manually felt that turning the record button on and off was “somewhat distracting” while users who applied the rules automatically in the background found it less distracting. One of the users who applied the rules manually described the switching between record on and off to be “tedious”. Since the tool offers both manual and background application, one way to address this issue is to have users simply use the manual application during the tool setup as a test mode and then apply them in the background when they have created an acceptable set of rules.

6.3 Link Usability: Ease of Accessing Artifacts

User feedback indicates that the captured links were usable to the participants. Most of the participants liked the ability to link design elements

to documentation, the ability to link to specific points within the documentation, and the ease of navigating to the documentation from design. One participant commented, "It could take me back to the exact location of the comments made in documentation files." Another participant stated that it is "easy to link & view artifact from tool itself. Jumps to edits automatically". Still another participant liked "linking to precise points in the document". One participant liked the ability to view the artifacts in their native tool: "Simplifies the process, everything is integrated into one workspace".

6.4 Tension between Automatic and User-Controlled Capture

There is a tension between the automated capture and user-controlled capture of links. Even though increasing user control requires more time from the user, some participants preferred the ability to manually map artifacts to design via "drag and drop". Two participants would like to be able to explicitly indicate the artifacts they are linking through a button in the native editor. One participant even suggested displaying a dialog box to manually confirm the links to add after each recording session: "I'd like to have more control over the links. I'd like to have checkboxes to manually pick the one link I wanted." One participant also wanted to be able to manually enter labels for the captured links.

On the other hand, some users prefer less user control and more automated capture. Three users disliked explicitly turning the record button on and off, with one user commenting "Start / stop recording button kinda distracting."

6.5 Privacy Concerns

Since capturing user interaction takes place in the background, participants were asked if they had privacy concerns over the logging of their interaction events. One participant stated that there was no privacy concern "if I can clearly see and select what is recorded. Otherwise, yes". Two participants stated that as long as the logging only takes place within the ACTS tool, they have no privacy concerns. However, once the logging goes outside the tool, e.g. logging visited sites on a browser, it may become a privacy issue. To address privacy concerns, one participant suggested, "Maybe allow me to specify (Black/White list) which apps are/are not recorded". Thus, as long as the recording is limited within the tool or is transparent, the participants are amenable to logging their user interactions.

6.6 Usability of the Tool

While the participants like the idea of being able to automatically capture links to documentation, they expressed several usability issues regarding the current tool implementation. Some participants would like more visual feedback on the status of link capture. For instance, they would like a better visual indicator that the recording is taking place. One user would like more "on-the-fly" directions to know what actions to take while recording. These usability issues can be addressed by further tool development.

Another usability issue is in the imposed actions that the participants must follow to indicate links between elements in the design and the related documentation. For instance, participants have to explicitly select an element in the design (via clicking or double-clicking a component) prior to opening a related artifact in order for the tool to create traceability links. In addition, participants must explicitly turn on the record button to start capturing links, must save the PDF file using an external button, and must open the files through ACTS in order to capture the links. Two participants felt so constrained by these requirements that one participant called the recording approach as "heavyweight" while another participant referred to the tool as "clunky". The tool limitations regarding link capture can be addressed by developing more sophisticated rules that can understand a wider range of user events. In addition, providing user preferences for recording as well as minimizing additional user required actions to capture the links can address the other usability issues.

Still another usability issue is in understanding the rules. The current representation of rules as XSLT is difficult to understand and create for an average user. One way to address this is by providing a form-based user interface that users can use to enter their rules. Some users would also like to see a visualization of how links are being transformed by the rules, or a preview of what the links would look like after the rule has been applied.

6.7 Discussion

The user feedback helps us understand some of the desirable features as well as concerns the users have with the current implementation of ACTS.

Based on user feedback, the overhead incurred in link capture seems to be generally acceptable to the participants. The features that most users liked about the tool are the automatic linking from the structural design to documentation and the automatic linking to specific locations within the documentation. Most users expressed usability issues with the current tool implementation.

As far as the capturing the links in the background, we received mixed results, as indicated by Section 6.4. It seemed that some users prefer more implicit capture while others prefer more explicit capture. For instance, some users disliked turning the record button on and off while some prefer more user interaction with the tool. The latter may be attributed to the following reasons. First, some users had a limited understanding of the capabilities of the rules. Secondly, some users did not want to lose the links they identified within the context of the tool. In the current implementation of ACTS, users may examine the links and interactively apply the rules after a recording session, which means that some time has elapsed and they are required to remember the context of the artifact. Thus, enabling users to interactively specify which artifacts to link during the recording session is also important.

While the participants found many usability issues with the tool, some of them indicated continued usage of the tool. Fifteen of the participants would like to use the tool in the future and an additional 5 would "maybe" use the tool if their task required linking documentation to design.

7 Conclusion

This paper examined the traceability challenges and showed that the complexity of the problem stems from multiple interacting factors: economic, technical, and social. The ACTS traceability framework begins to tackle these challenges by unifying distributed and varied artifacts around the architecture, by supporting stakeholder customization, and by prospectively capturing links. This paper discussed the technical challenges involved in tracing across heterogeneous artifacts at different levels of granularity, at integrating third party tools into a traceability system, at maintaining traceability links, and at decoupling the heuristics from the underlying trace mechanisms. Our traceability framework is extensible and can integrate existing trace search techniques. We have demonstrated the technical feasibility of our approach through a case study, an exemplar implementation, and user feedback.

Further work is needed in relating different types of architectural models such as behavioral and interaction models and in understanding the effectiveness of the notification adapter. In addition, more work is needed to understand the social and economic implications of the ACTS framework. For instance, it is important to understand how to balance the cost of creating custom rules versus achieving better precision/recall rates. It is also important to understand how to balance individual versus organizational priorities in capturing traceability links. Another open research topic is minimizing the cost while scaling the approach to hundreds or perhaps thousands of links.

8 Acknowledgment

The authors are grateful to Gloria Mark for providing guidance on the user study. The authors would like to thank S. Hendrickson for ArchStudio support, A. Baquero, E. Dashofy, K. Strasser, N. Takeo, Y. Zheng, and other ArchStudio users for feedback, and S. Cutler, D. Kwok, C. Leu, A. Marron, J. Meevasin, H. Pham, D. Purpura, and A. Rahnemoon for tool development. This research has been supported by grants from the National Science Foundation IIS-0808783, CCF-0917129, and the Naval Postgraduate School, Acquisition Research Program, through grant N00244-10-1-0038. No review, approval, or endorsement implied.

9 References

[1] IBM Rational Requisite Pro. http://www-01.ibm.com/software/awdtools/reqpro/.

[2] The Jazz Project. http://jazz.net.

[3] Resource description framework (RDF): Concepts and abstract syntax. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[4] Topic maps. http://www.topicmap.com/.

[5] SQLite. http://www.sqlite.org/, 2009.

[6] Adobe Systems Incroporated. Adobe Acrobat. http://www.adobe.com/products/acrobat/, 2009.

[7] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni. Model traceability. *IBM Systems Journal*, 45(3):515–26, 2006.

[8] J. Aldrich, C. Chambers, and D. Notkin. ArchJava: Connecting software architecture to implementation. In *Proc of the 24th Int'l Conference on Software Engineering*, pages 187–197, Orlando, FL, 2002.

[9] Ian Alexander. Towards automatic traceability in industrial practice. In *Proc of the 1st Int'l Workshop on Traceability*, pages 26–31, Edinburgh, September 2002.
[10] Joao Paulo Almeida, Pascal van Eck, and Maria-Eugenia Iacob. Requirements traceability and transformation conformance in model-driven development. In *Proc of the 10th Int'l Enterprise Distributed Object Computing Conference*, Hong Kong, 2006.
[11] Thomas A. Alspaugh, Hazeline U. Asuncion, and Walt Scacchi. Analyzing software licenses in open architecture software systems. In *2nd Int'l Workshop on Emerging Trends in Free/Libre/Open Source Software (FLOSS) Research and Development*, pages 54–57, Vancouver, 2009.
[12] Thomas A. Alspaugh, Hazeline U. Asuncion, and Walt Scacchi. Intellectual property rights requirements for heterogeneously-licensed systems. In *Proc of the 17th Int'l Requirements Engineering Conference*, Atlanta, 2009.
[13] Kenneth M. Anderson, Susanne A. Sherba, and William V. Lepthien. Towards large-scale information integration. In *Proc of the Int'l Conf on Software Engineering*, Orlando, 2002.
[14] Kenneth M. Anderson, Richard N. Taylor, and E. James Jr. Whitehead. A critique of the open hypermedia protocol. *Journal of Digital Information (JoDI)*, 1(2), 1997.
[15] Kenneth M. Anderson, Richard N. Taylor, and E. James Jr. Whitehead. Chimera: Hypermedia for heterogeneous software development environments. *ACM Trans on Information Systems*, 18(3):211–245, July 2000.
[16] Luis Filipe Andrade and José Luiz Fiadeiro. *Architecture Based Evolution of Software Systems*, volume 2804/2003 of *Formal Methods for Software Architectures - Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2003.
[17] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, 2002.
[18] Apache Software Foundation. Lucene. http://lucene.apache.org, 2006.
[19] Apache Software Foundation. Xalan-Java. http://xml.apache.org/xalan-j/, 2007.
[20] Paul Arkley and Steve Riddle. Overcoming the traceability benefit problem. In *Proc of the 13th Int'l Conference on Requirements Engineering*, pages 385–389, Paris, 2005.
[21] Paul Arkley and Steve Riddle. Tailoring traceability information to business needs. In *Proc of the 14th Int'l Requirements Engineering Conference*, Minneapolis, St. Paul, MN, 2006.
[22] Ove Armbrust, Alexis Ocampo, Jürgen Münch, Masafumi Katahira, Yumi Koishi, and Yuko Miyamoto. Establishing and maintaining traceability between large aerospace process standards. In *Proc of the 5th Int'l Workshop on Traceability in Emerging Forms of Software Engineering*, pages 36–40, Vancouver, Canada, May 2009.
[23] Hazeline Asuncion. *Architecture-Centric Traceability for Stakeholders (ACTS)*. Ph.D. Thesis (Info & Computer Science), UC, Irvine, 2009.
[24] Hazeline Asuncion, Arthur Asuncion, and Richard N. Taylor. Software traceability with topic modeling. In *Proc of 32nd Int'l Conference on Software Engineering*, Cape Town, 2010.
[25] Hazeline Asuncion, Frédéric François, and Richard N. Taylor. An end-to-end industrial software traceability tool. In *Proc of the 6th Joint Meeting of the European Software Eng Conf and the ACM SIGSOFT Int'l Symp on the Foundations of Software Engineering (ESEC/FSE)*, Dubrovnik, 2007.
[26] Hazeline Asuncion and Richard N. Taylor. Capturing custom link semantics among heterogeneous artifacts and tools. In *5th Int'l Workshop on Traceability in Emerging Forms of Software Engineering*, Vancouver, British Columbia, 2009.
[27] Roberto Basili, Maria Teresa Pazienza, and Fabio Massimo Zanzotto. Inducing hyperlinking rules in text collections. In *Recent Advances in Natural Language Processing*, pages 131–140, Borovets, Bulgaria, 2003.
[28] Steve Berczuk, Brad Appleton, and Robert Cowham. The trouble with tracing: Traceability dissected. http://www.cmcrossroads.com/content/view/6685/264/, May 2005.
[29] Alessandro Bianchi, Anna Rita Fasolino, and Guiseppe Visaggio. An exploratory case study of the maintenance effectiveness of traceability models. In *Proc of the 8th Int'l Workshop on Program Comprehension (IWPC)*, page 149, Limerick, Ireland, 2000.
[30] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley Object Technology Series. Reading, Massachusetts, Addison-Wesley Professional, 2nd edition, 2005.
[31] Jan Bosch. Architecture-centric software engineering. In *Proc of the 24th Int'l Conference on Software Engineering*, pages 681–682, Orlando, 2002.
[32] José A. Camacho-Guerrero, Alex A. Carvalho, Maria G. C. Pimentel, Ethan V. Munson, and Alessandra A. Macedo. Clustering as an approach to support the automatic definition of semantic hyperlinks. In *Proc of the 18th Conf on Hypertext and Hypermedia*, Manchester, 2007.
[33] John M. Carroll and Mary Beth Rosson. Deliberated evolution: Stalking the view matcher in design space. In *Design Rationale: Concepts, Techniques, and Use*, pages 107–145. Lawrence Erlbaum Associates, Inc., 1996.
[34] Shang-Wen Cheng, David Garlan, Bradley Schmerl, JoÃ£o Pedro Sousa, Bridget Spitznagel, Peter Steenkiste, and Ningning Hu. Software architecture-based adaptation for pervasive systems. In *Int'l Conference on Architecture of Computing Systems (ARCS 2002)*, Karlsruhe, Germany, 2002.
[35] Jane Cleland-Huang. Toward improved traceability of non-functional requirements. In *Proc of the 3rd Int'l Workshop on Traceability in Emerging Forms of Software Engineering*, pages 14–19, Long Beach, CA, 2005.
[36] Jane Cleland-Huang. Just enough requirements traceability. In *Proc of the 30th Annual Int'l Computer Software and Applications Conference (COMPSAC)*, Chicago, 2006.
[37] Jane Cleland-Huang, Carl K. Chang, and Mark Christensen. Event-based traceability for managing evolutionary change. *IEEE Transactions on Software Engineering*, 29(9):796–810, 2003.
[38] Jane Cleland-Huang and Rafal Habrat. Visual support in automated tracing. In *2nd Int'l Workshop on Requirements Engineering Visualization*, New Delhi, 2007.
[39] Jane Cleland-Huang, Raffaella Settimi, Eli Romanova, Brian Berenbach, and Stephen Clark. Best practices for automated traceability. *Computer*, 40(6):27–35, June 2007.
[40] Jane Cleland-Huang, Grant Zemont, and Wiktor Lukasik. A heterogeneous solution for improving the return on investment of requirements traceability. In *Proc of the 12th Int'l Requirements Engineering Conf*, Kyoto, 2004.
[41] Jeff Conklin and Michael L. Begeman. gIBIS: A hypertext tool for exploratory policy discussion. *ACM Transactions on Information Systems (TOIS)*, 6(4):303–331, 1988.
[42] Davor Cubranic, Gail C. Murphy, Janice Singer, and S. Booth Kellogg. Hipikat: a project memory for software development. *ÍEEE Transactions on Software Engineering*, 31(6):446–65, 2005.
[43] Eric Dashofy. *Supporting Stakeholder-driven, Multi-view Software Architecture Modeling*. Ph.D. Thesis, University of California, Irvine, 2007.
[44] Eric Dashofy, André van der Hoek, and Richard N. Taylor. A comprehensive approach for the development of XML-based software architecture description languages. *ACM Trans on Software Eng and Methodology*, 14(2):199–245, 2005.
[45] Eric M. Dashofy, Hazel Asuncion, Scott A. Hendrickson, Girish Suryanarayana, John C. Georgas, and Richard N. Taylor. Archstudio 4: An architecture-based meta-modeling environment. In *Proc of the 29th Int'l Conference on Software Engineering (ICSE 2007)*, volume Informal Research Demonstrations, pages 67–68, Minneapolis, 2007.
[46] Cleidson R. B. de Souza, T. Hildenbrand, and David Redmiles. Toward visualization and analysis of traceability relationships in distributed and offshore software development projects. In *First Int'l Conference on Software Engineering Approaches for Offshore and Outsourced Development (SEAFOOD)*, Zurich, Switzerland, 2007.
[47] Robert DeLine, Mary Czerwinski, and George Robertson. Easing program comprehension by sharing navigation data. In *Proc of the 2005 Symposium on Visual Languages and Human-Centric Computing*, pages 241–248, September 2005.
[48] Ralf Dömges and Klaus Pohl. Adapting traceability environments to project specific needs. *Communications of the ACM*, 41(12):54–62, 1998.
[49] Chuan Duan and Jane Cleland-Huang. Visualization and analysis in automated trace retrieval. In *Proc of the 1st Int'l Workshop on Requirements Engineering Visualization*, Minneapolis/St. Paul, 2006.

[50] Chuan Duan and Jane Cleland-Huang. Clustering support for automated tracing. In *Proc of the 22nd Int'l Conference on Automated Software Engineering*, pages 244–253, Atlanta, 2007.
[51] Edgewall. Trac open source project: Integrated scm & project management. http://trac.edgewall.org/.
[52] Alexander Egyed, Stefan Biffl, Matthias Heindl, and Paul Grünbacher. Determining the cost-quality trade-off for automated software traceability. In *Proc of the 20th Int'l Conference on Automated Software Engineering*, pages 360–363, Long Beach, CA, 2005.
[53] Alexander Egyed, Stefan Biffl, Matthias Heindl, and Paul Grünbacher. A value-based approach for understanding cost-benefit trade-offs during automated software traceability. In *Proc of the 3rd Int'l Workshop on Traceability in Emerging Forms of Software Engineering*, Long Beach, CA, 2005.
[54] Alexander Egyed and Paul Grünbacher. Automating requirements traceability: Beyond the record & replay paradigm. In *Proc of the 17th Int'l Conf on Automated Software Engineering*, pages 163–171, Edinburgh, Scotland, 2002.
[55] Michael Evans. SPMN director identifies 16 critical software practices. *CrossTalk, The Journal of Defense Software Engineering*, March 2001.
[56] Peter H. Feiler, Bruce Lewis, and Stephen Vestal. The SAE avionics architecture description language (AADL) standard: A basis for model-based architecture-driven embedded systems engineering. In *RTAS 2003 Workshop on Model-Driven Embedded Systems*, Washington, D.C., 2003.
[57] Roy Fielding, E. James Whitehead, Kenneth Anderson, Peyman Oreizy, Gregory A. Bolcer, Peyman Oreizy, and Richard N. Taylor. Web-based development of complex information products. *Communications of the ACM*, 41(8):84–92, August 1998.
[58] David Garlan, Robert T. Monroe, and David Wile. Acme: An architecture description interchange language. In *CASCON '97*, pages 169–183, Toronto, Ontario, Canada, 1997.
[59] John C. Georgas, Eric M. Dashofy, and Richard N. Taylor. Architecture-centric development: A different approach to software engineering. *ACM Crossroads, issue on Software Engineering*, 12(4), 2006.
[60] Mark J. Gerken, Nancy A. Roberts, and Douglas A. White. The knowledge-based software assistant: A formal, object oriented software development environment. In *Proc of the National Aerospace and Electronics Conference*, volume 2, pages 511–18, Dayton, OH, 1996.
[61] Daniel M. German and Ahmed E. Hassan. License integration patterns: Addressing license mismatches in component-based development. In *Proc of the Int'l Conference on Software Engineering*, pages 188–198, Vancouver, Canada, 2009.
[62] Martins Gills. Survey of traceability models in IT projects. In *Proc of ECMDA Traceability Workshop (ECMDA-TW)*, 2005.
[63] Max. Goldman and Robert C. Miller. Codetrail: Connecting source code and web resources. In *Visual Languages and Human-Centric Comp*, Herrsching am Ammersee, 2008.
[64] O.C.Z. Gotel and C.W. Finkelstein. An analysis of the requirements traceability problem. In *Proc of 1st Int'l Conf on Requirements Engineering*, Colorado Springs, 1994.
[65] Orlena Gotel and Anthony Finkelstein. Modeling the contribution structure underlying requirements. In *1st Int'l Workshop on Requirements Engineering: Foundations for Software Quality*, 1994.
[66] Mark Grechanik, Kathryn S. McKinley, and Dewayne E. Perry. Recovering and using use-case-diagram-to-source-code traceability links. In *Proc of 6th Joint Meeting of the ESEC/FSE*, Dubrovnik, 2007.
[67] J. Gustafsson, J. Paakki, L. Nenonen, and A.I. Verkamo. Architecture-centric software evolution by software metrics and design patterns. In *Proc of the Sixth European Conference on Software Maintenance and Reengineering*, pages 108–115, Budapest, Hungary, March 2002.
[68] J. H. Hayes and A. Dekhtyar. Humans in the traceability loop: Can't live with 'em, can't live without 'em. In *Proc of the 3rd Int'l Workshop on Traceability in Emerging Forms of Software Engineering*, pages 20–23, Long Beach, CA, 2005.
[69] Jane Hayes and Alex Dekhtyar. Grand challenges for traceability. Technical Report COET-GCT-06-01-0.9, Center of Excellence for Traceability, http://www.traceabilitycenter.org/files/COET-GCT-06-01-0.9.pdf, 2007.
[70] Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Transactions in Software Engineering*, 32(1):4–19, 2006.
[71] Jane Huffman Hayes, Alex Dekhtyar, Senthil Karthikeyan Sundaram, and Sarah Howard. Helping analysts trace requirements: An objective look. In *Proc of the 12th Int'l Requirements Engineering Conference*, pages 249–259, Kyoto, Japan, 2004.
[72] Scott A. Hendrickson and André van der Hoek. Modeling product line architectures through change sets and relationships. In *Proc of the 29th Int'l Conference on Software Engineering (ICSE 2007)*, pages 189–198, Minneapolis, MN, 2007.
[73] William C. Hill, James D. Hollan, Dave Wroblewski, and Tim McCandless. Edit wear and read wear. In *SIGCHI Conference on Human Factors in Computing Systems*, Monterey, California, 1992.
[74] John Horner and Michael E. Atwood. Design rationale: the rationale and the barriers. In *Proc of the 4th Nordic Conference on Human-Computer Interaction: Changing Roles*, Oslo, Norway, 2006.
[75] P Inverardi, H. Muccini, and Patrizio Pelliccione. Automated check of architectural models consistency using SPIN. In *Proc of the 16th Int'l Conference on Automated Software Engineering*, page 346, San Diego, CA, 2001.
[76] Matthias Jarke. Requirements tracing. *Communications of the ACM*, 41(12):32–36, 1998.
[77] Frédéric Jouault. Loosely coupled traceability for ATL. In *Proc of the European Conference on Model Driven Architecture (ECMDA) Workshop on Traceability*, pages 29–37, Nuremberg, Germany, 2005.
[78] Huzefa Kagdi, Jonathan I. Maletic, and Bonita Sharif. Mining software repositories for traceability links. In *Proc of the 15th IEEE Int'l Conf on Program Comprehension*, 2007.
[79] Mik Kersten and Gail C. Murphy. Mylar: A degree-of-interest model for IDEs. In *Proc of 4th Int'l Conf on Aspect-oriented Software Development*, Chicago, 2005.
[80] Jintae Lee. Extending the potts and bruns model for recording design rationale. In *Proc of the 13th Int'l Conference on Software Engineering*, pages 114–125, Austin, 1991.
[81] Jörg Leuser. Challenges for semi-automatic trace recovery in the automotive domain. In *Proc of the 5th Int'l Workshop on Traceability in Emerging Forms of Software Engineering*, pages 31–35, Vancouver, Canada, 2009.
[82] Mikael Lindvall and Kristian Sandahl. Practical implications of traceability. *Software - Practice and Experience*, 26(10):1161–80, 1996.
[83] Allan MacLean, Richard M. Young, Victoria M. E. Bellotti, and Thomas P. Moran. Questions, options, and criteria: Elements of design space analysis. In *Design Rationale: Concepts, Techniques, and Use*, pages 53–105. Lawrence Erlbaum Associates, Inc., 1996.
[84] Andrian Marcus and Jonathan I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proc of the 25th International Conference on Software Engineering*, Portland, 2003.
[85] Nenad Medvidovic and Richard N. Taylor. A framework for classifying and comparing architecture description languages. In *Proc of the 6th European Software Engineering Conference/5th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 60–76, Zurich, Switzerland, 1997.
[86] Mozilla. XUL. https://developer.mozilla.org/en/XUL, 2009.
[87] Ethan V. Munson. The software concordance: Bringing hypermedia to software development environments. In *Brazilian Symp on Multimedia and Hypermedia Systems*, Goiania, 1999.
[88] L.G.P. Murta, André van der Hoek, and C.M.L. Werner. ArchTrace: Policy-based support for managing evolving architecture-to-implementation traceability links. In *Proc of the 21st Int'l Conference on Automated Software Engineering (ASE 2006)*, pages 135–144, Tokyo, Japan, 2006.
[89] Elena Navarro. *ATRIUM: Architecture Traced from RequIrements by applying a Unified Methodology*. Ph.D. Thesis, University of Castilla-La Mancha, 2007.
[90] Christian Neumüller and Paul Grünbacher. Automating software traceability in very small companies - a case study and lessons learned. In *Proc of the 21st Int'l Conference on Automated Software Engineering*, pages 145–156, Tokyo, Japan, 2006.
[91] Bashar Nuseibeh. Weaving together requirements and architectures. *Computer*, 34(3):115–117, March 2001.
[92] Rob van Ommering, Frank van der Linden, Jeff Kramer, and Jeff Magee. The koala component model for consumer electronics software. *IEEE Computer*, 33(3):78–85, 2000.
[93] Open Source Initiative. http://www.opensource.org/, 2008.
[94] Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Greg Johnson, Nenad Medvidovic, Alex Quilici, David S.

Rosenblum, and Alexander L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62, 1999.
[95] Peyman Oreizy, Nenad Medvidovic, and Richard N. Taylor. Architecture-based runtime software evolution. In *Proc of the 20th Int'l Conference on Software Engineering (ICSE '98)*, pages 177–186, Kyoto, Japan, 1998.
[96] Kasper Osterbye and Uffe Kock Wiil. The flag taxonomy of open hypermedia systems. In *Proc of the 7th Conference on Hypertext and Hypermedia*, pages 129–39, Washington, DC, March 1996.
[97] Francisco A. C. Pinheiro and Joseph A. Goguen. An object-oriented tool for tracing requirements. *IEEE Software*, 13(2):52–64, 1996.
[98] Klaus Pohl. PRO-ART: Enabling requirements pre-traceability. In *Proc of 2nd Int'l Conf on RE*, Colorado Springs, 1996.
[99] Klaus Pohl. *Process-Centered Requirements Engineering*. John Wiley & Sons, Inc., New York, NY, 1996.
[100] Klaus Pohl, Mathias Brandenburg, and Alexander Gülich. Integrating requirement and architecture information: A scenario and meta-model based approach. In *Proc of the 7th Int'l Workshop on Requirements Engineering: Foundation for Software Quality*, pages 68–84, Interlaken, Switzerland, June 2001.
[101] Klaus Pohl and Stephan Jacobs. Enabling traceability and mutual understanding. *Concurrent Engineering: Research and Applications*, 2(4):279–90, 1994.
[102] Progress Software Corporation. XSLT Editor. http://www.stylusstudio.com/, 2009.
[103] B. Ramesh, T. Powers, C. Stubbs, and M. Edwards. Implementing requirements traceability: A case study. In *Proc of the 1995 Int'l Symp on Requirements Engineering*, York, 1995.
[104] Bala Ramesh and Matthias Jarke. Towards reference models for requirements traceability. *IEEE Transactions in Software Engineering*, 27(1):58–93, 2001.
[105] Julian Richardson and Jeff Green. Automating traceability for generated software artifacts. In *Proc of the 19th Int'l Conf on Automated Software Engineering*, Linz, 2004.
[106] Till Schummer. Lost and found in software space. In *34th Annual Hawaii Int'l Conference on System Sciences*, 2001.
[107] Janice Singer, Robert Elves, and Margaret-Anne Storey. Navtracks: Supporting navigation in software maintenance. In *Proc of the 21st Int'l Conf on Software Maint*, Budapest, 2005.
[108] Margaret E. Singleton. *Automating Code and Documentation Management*. Prentice-Hall, Inc., Upper Saddle River, NJ, 1986.
[109] G. Spanoudakis, A. Zisman, E. Perez-Miñana, and P. Krause. Rule-based generation of requirements traceability relations. *Journal of Systems and Software*, 72(2):105–27, July 2004.
[110] George Spanoudakis and Andrea Zisman. *Software Traceability: A Roadmap*, volume 3 of *Handbook of Software Engineering and Knowledge Engineering*. World Scientific Publishing, 2005.
[111] John Spencer. Architecture description markup language (adml): Creating an open market for it architecture tools. White Paper, The Open Group, September 26 2000.
[112] R. C. Sugden and M. R. Strens. Strategies, tactics and methods for handling change. In *Proc of the IEEE Symposium and Workshop on Engineering of Computer-Based Systems*, Friedrichshafen, Germany, March 1996.
[113] SyncRO Soft Ltd. Oxygen XSL / XSLT editor. http://www.oxygenxml.com/, 2009.
[114] Antony Tang, Muhammad Ali Babar, Ian Gorton, and Jun Han. A survey of the use and documentation of architecture design rationale. In *Proc of the 5th Working IEEE/IFIP Conference on Software Architecture*, pages 89–98, Pittsburgh, PA, 2005.
[115] R. N. Taylor, N. Medvidovic, and E.M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, 2010.
[116] Timo Tuunanen, Jussi Koskinen, and Tommi Kärkkäinen. Automated software license analysis. *Automated Software Engineering*, 16(3-4):455–490, 2009.
[117] Unity Technologies. End user license agreement. http://unity3d.com/unity/unity-end-user-license-2.x.html, 2008.
[118] K. Ven and H. Mannaert. Challenges and strategies in the use of Open Source Software by Independent Software Vendors. *Information and Software Technology*, 50(9-10):991–1002, 2008.
[119] M. Vieira, M. Dias, and D.J. Richardson. Analyzing software architectures based on statechart semantics. In *Proc 21st Int'l Conference on Software Engineering (ICSE 21)*, Limerick, Ireland, 2000.
[120] Antje von Knethen and Barbara Paech. A survey on tracing approaches in practice and research. Tech Report IESE-Report Nr. 095.01/E, Fraunhofer Institut Experimentelles Software Engineering, Fraunhofer Gesellschaft, 2002.
[121] Anke Weber, Holger M. Kienle, and Hausi A. Muller. Live documents with contextual, data-driven information components. In *Proc of the 20th Annual Int'l Conference on Computer Documentation*, 2002.
[122] Alan Wexelblat and Pattie Maes. Footprints: History-rich tools for information foraging. In *Proc of the SIGCHI Conf on Human Factors in Computing Systems*, Pittsburgh, 1999.
[123] Uffe K. Wiil and John J. Leggett. The hyperdisco approach to open hypermedia systems. In *Proc of the 7th Conference on Hypertext*, pages 140–148, Bethesda, MD, 1996.
[124] Lihua Xu, Scott A. Hendrickson, Eric Hettwer, Hadar Ziv, André van der Hoek, and Debra J. Richardson. Towards supporting the architecture design process through evaluation of design alternatives. In *Proc of the 2nd Int'l Workshop on the Role of Software Architecture in Testing and Analysis (ROSATEA 2006)*, 2006.
[125] Xin Zhou, Zhenzhong Huo, Yaowen Huang, and Jian Xu. Facilitating software traceability understanding with ENVISION. In *32nd Int'l Computer Software and Applications*, pages 295–302, Turku, 2008.
[126] Lijie Zou, Michael W. Godfrey, and Ahmed E. Hassan. Detecting interaction coupling from task interaction histories. In *Proc of the 15th Int'l Conference on Program Comprehension*, 2007.

Hazeline U. Asuncion is an Assistant Professor at the Computing and Software Systems Program, University of Washington, Bothell. She was previously a Postdoctoral Researcher at the Institute for Software Research at the University of California, Irvine. She has worked in the industry in a variety of roles: as a software engineer at Unisys Corporation, and as a traceability engineer at Wonderware Corporation where she designed a successful in-house traceability system. Her research emphasis is on software traceability. She has also investigated the tracing of software license conflicts in heterogeneously composed software systems.

Richard N. Taylor is a Professor of Information and Computer Sciences at the University of California at Irvine and Director of the Institute for Software Research. He received the Ph.D. degree in Computer Science from the University of Colorado at Boulder in 1980. His research interests are centered on design and software architectures, especially event-based and peer-to-peer systems. He was recognized as an ACM Fellow in 1998. In 2005 he was awarded the ACM SIGSOFT Distinguished Service Award and in 2009 he was recognized with the 2009 ACM SIGSOFT Outstanding Research Award.