Software Agent Architecture for Consistency Management in Distributed Documents

Anthony Finkelstein, Danila Smolko
University College London
Department of Computer Science
Gower Street - London WC1E 6BT - UK
Phone: 44 207 6797293 Fax: 44 207 3871397

{A.Finkelstein | D.Smolko}@cs.ucl.ac.uk

ABSTRACT

We give an account of an architecture for management of consistency relations between distributed documents. We propose a collaborative framework of interacting software agents, capable of concurrent execution of consistency checks and evaluate architectural performance and scalability. We demonstrate the advantages of use of this system in a domain of software engineering documents.

Keywords

Consistency management, XML, software agents, co-operation, distributed architecture.

1. INTRODUCTION

Co-operation between physically distributed components or actors in a collaborative activity is becoming commonplace [2]. Particularly important is the ability to check and maintain consistency relations between collaboratively authored distributed resources throughout their development cycle. The heterogeneity of products being developed and the multiplicity of stakeholders and development participants may generate inconsistencies and conflicts in the distributed documents [7]. The software agent architecture for consistency management aims to provide a solution in the form of an infrastructure to carry out consistency checks in a distributed setting.

The architecture spans two interacting levels. Co-operation and interaction [8] between agents in exchange of information is the base architectural level, on which the functional level – a consistency management framework is built. The development of the consistency level is a team effort of the authors and their colleagues, and is described in detail elsewhere [4][9]. The goal of current research is to enhance the existing consistency management framework by developing a base interaction level between the distributed framework components, thus enabling the resulting architecture to effectively and efficiently provide consistency management service of heterogeneous distributed resources. We also seek to evaluate the potential of use of mobile agents for consistency management applications.

2. ARCHITECTURE DESCRIPTION

Software Agent Co-operation Level

In design of the base level of architecture, we explicitly aimed at building a distributed and de-centralised solution, where all operations with respect to provided services are performed at the network host, where such services are required. The resulting system will then capitalise on effective use of distribution to avoid bottlenecks and achieve scalability with an increase in a number of transactions.

The architecture is schematically depicted in Fig. 1, which shows architectural components. A replicated central domain server serves the purpose of providing support to the agents in locating all required resources from the destination host, where they operate. The domain server is accessible from every network host, participating in the system, and hosts a resource and agent location database, thereby providing location transparency for all participants of the system.

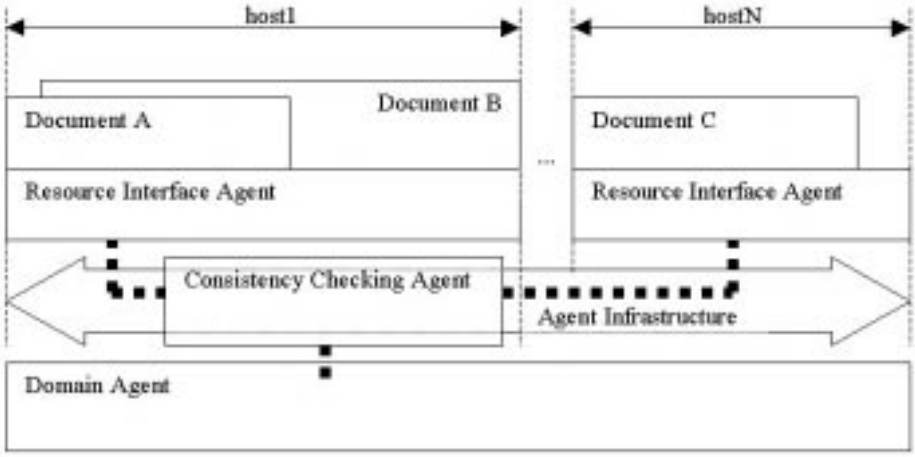


Fig. 1. Software Agent Architecture for Distributed Consistency Management

In aiming to build a re-active and persistent architecture [6], we have designed an event-driven system from the start. Major events are caused by changes of existing resources, inclusion, exclusion and merger of resources. History record of these events is kept at resource locations and in a dedicated database at the domain server. Most such events trigger a single or a group of concurrently executed consistency checks. Support for resource history records allows us to seamlessly co-ordinate and integrate heterogeneous concurrent checks to achieve efficient collaboration.

Agent-to-agent collaboration

One of important advantages of use of this distributed architecture is support for collaborative checking of related consistency rules. When related documents are distributed, the majority of time spent in a consistency check results from the need to retrieve various document parameters across the network.

In the proposed approach, consistency checking agents carry all the data they collect with them. Thus, already collected information could be shared between agents, increasing overall efficiency of the system.

As each consistency relation affects numerous documents, a consistency check of this relationship can be originated from more than one of these documents at the same time, thus leading to a number of agents checking the same consistency rule concurrently.

The registration operation of the mobile consistency checking agent at the domain is aimed at resolving this problem.

During registration, the agent deposits a reference to the consistency rule being checked, together with URLs of documents that have been checked already and values of retrieved parameters into an agent repository. After the registration, the agent is able to scan the repository references to a similar consistency rule. If such references are found, then one or more *redundant* consistency checking agents have been identified.

Redundant agents then have an opportunity to cross-check and update their collected data, and terminate all instances of themselves but one. This approach ensures that at any given time, at any host on the network, there is no more than one consistency checking agent working on any of the numerous consistency rules in the system.

After redundancy issues are resolved, then a search commences for agents, which require the kind of information that this particular agent has collected and is able to provide. This is accomplished by scanning the repository for consistency rules, relating to the elements of a certain document type, being checked by this particular agent. When such rules are found, the required values may be exchanged.

Consistency Management Level

This level provides a framework to express and check consistency relations between the documents, which have "overlapping" content. Within the software engineering domain, related UML designs of a project could serve as a good example of such documents [4][9]. Relationships between collaboration diagrams and class diagram arise from an “overlap” between these documents.

The consistency framework aims to cross-check heterogeneous structured documents of different formats. To simplify design and respect this heterogeneity requirement, eXtensible Markup Language (XML) [1] has been adopted as a intermediary format, which would both preserve data structure, and provide an effective method of access to the data by parsing XML and traversing the resulting XML tree. Our assumption is that it is possible to build a Document Type Definition (DTD), specifying a sequence of structural elements, for each document type used in the system.

Consistency Rules

The consistency framework is centred on the notion of a *consistency rule*, which expresses a desirable relation between two or more types of "overlapping" documents. Each rule includes the pointers to related elements within data structures of respective document types, presented as XPointer [3] expressions. The consistency relation in a rule is specified within the set of allowed operations: equality, comparisons and

arithmetic operations being some of the most commonly used ones. Most recent work [11] provides a developing classification of consistency relations, which can be expressed by use of the proposed framework.

Transparency and Locality of Access to Distributed Documents

By using XLink pointers to documents and XPointer expressions for access to document elements, the architecture supports distributed location of documents and provides transparency of distribution for the documents and ease of access to their elements. Once a consistency check is taking place, the physical document locations of related documents specified the consistency rule are resolved with the document name tables located at domains and at the global gateway domain. Mobility allows the checking agent to migrate to the hosts, where source and destination documents are located, and perform consistency checks on these documents *locally*.

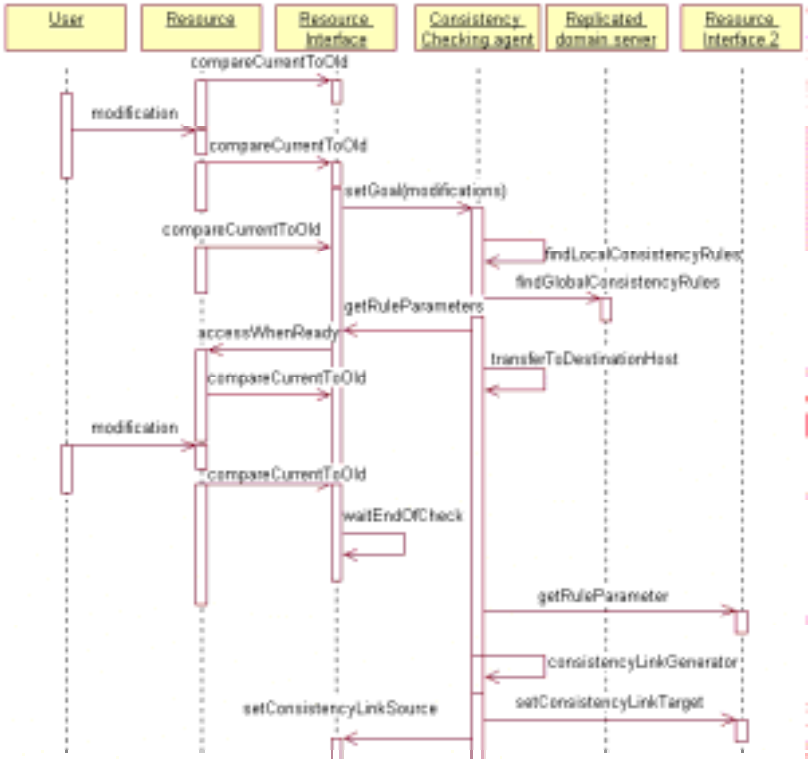


Fig. 2. Sequence diagram of interactions between components of the two architecture layers

Local access to documents gives a number of advantages in terms of access efficiency and is also a natural way to perform a consistency check in the case where documents have complex structure and multiple document elements are being concurrently checked. Another advantage of local consistency checks lies in assurance that the entire contents of documents are never transmitted across the network, only the values of particular elements may be transferred within the agent's data array. Thus, the approach ensures that security is maintained evenly across the whole distributed document system, at the level supported by the mobile agent's underlying implementation.

The result of a consistency check is computed when the operation stated in the consistency rule is applied to the values of constrained document elements, specified by the rule. The consistency checker mobile agent compares the values specified by the rule and computes the result, showing whether the relationship between documents is consistent or not. It then creates a consistency link between the documents. Such links are stored in the history file, and can be used to navigate from one document to the other, for instance when inconsistencies are being corrected.

Interactions with the Software Agent Level

The sequence diagram in Fig. 2 depicts interactions between actors of the two architectural levels. Resource interfaces, replicated domain agent and consistency checking agent are actors of the base architectural level. Resource, user of the resource and consistency checking functionality of the consistency agent belong to the consistency management level.

Interactions between the levels are driven by events, which are thrown and handled by system components. For example, the resource interface at regular time intervals throws an event *compareCurrentToOld*. This event is handled by its own method within the resource interface, which performs a comparison of a backup copy of the resource with its current version. The result - a tree-wise difference between these two XML documents – allows us to identify incremental changes (deletion, addition of document elements and changes of element values). If some changes appear to have been made by a user, they are marked up in XML and thrown as a *setGoal* event.

SetGoal event handler creates a new instance of a consistency checking agent and passes the list of changes to the agent as a parameter. The agent must then select those consistency rules, which are concerned with consistency relations of changed elements. Two rule databases need to be searched: local rules at resource host and global rule base at the domain server. Events *findLocalConsistency Rules* and *findGlobalConsistencyRules* result in a complete list of relevant rules being composed for the consistency agent.

The consistency management level of the architecture becomes actively involved at the next phase – carrying out of the consistency check. Each of consistency rules to be checked need to be filled in as "templates" with the values of document parameters. *GetRuleParameters* event results in the source XML document being parsed and values of document parameters, specified by each rules, being extracted and passed to the consistency checking agent.

As consistency rules involve two or more documents, the consistency agent identifies location of related, "target" documents. *TransferToDestinationHost* event is handled by the agent migrating to the host, where the next target document is stored.

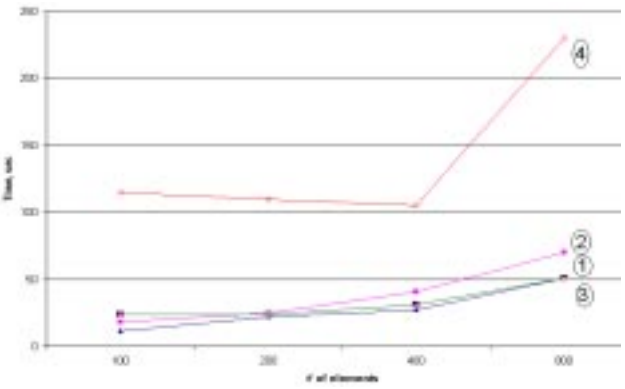
At each "target" host, the consistency agent follows the same sequence of events: *getRuleParameters* and *transferToDestination Host*. When all required parameters for a consistency rule have been collected, the agent calls *consistencyLinkGenerator*, which generates a link file, connecting consistent and inconsistent documents through XLinks. *SetConsistencyLinkTarget* and *setConsistency LinkSource* event handlers perform actual modifications to the links within the history file at target and source hosts.

Performance Evaluation of the Architecture

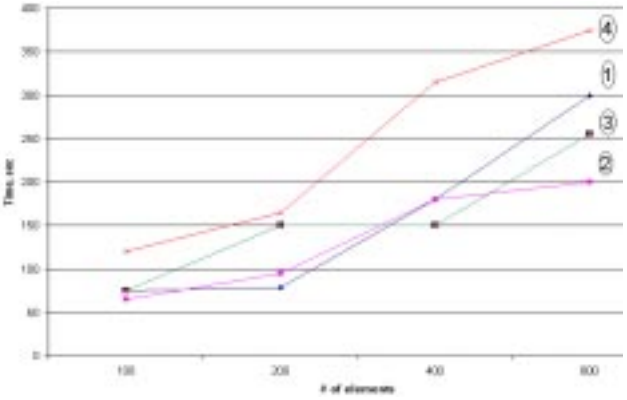
Performance evaluation is based on a prototype consistency management system, built on the proposed architecture. Consistency link generator, interface to consistency rules and retrieval of XML document parameters [4] are written in Java. IBM Aglets [5] are used as a mobile agent infrastructure. Consistency checking agent, resource interface and domain servers are based on the Aglets framework and use the messaging service provided by this framework for communication and event processing.

The documents used for evaluation of consistency checking performance are XML representations of UML collaboration and class diagrams, taken from the scheduling scenario [4][9][11]. These documents were replicated to emulate larger document sets where necessary. In building of the evaluation benchmarks, resource control techniques were followed [10], which allowed the system to provide consistency management services across a relatively large number of documents.

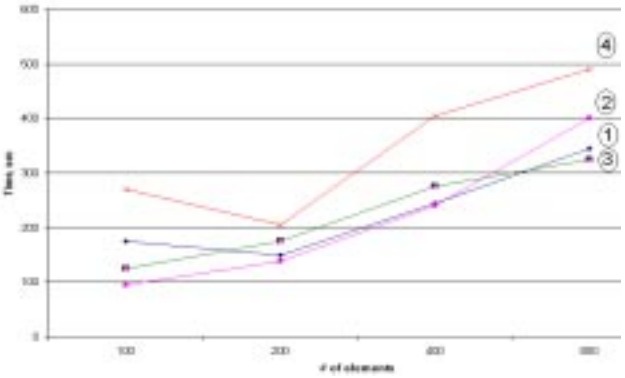
Fig. 3 depicts performance measurement graphs of the agent-based architecture. The data is collected in an experiment, where a number of distributed XML documents (1..50) are affected by a change in elements (1..50 changes out of 100..800 elements per document). These changes trigger selection of applicable consistency rules (1..20 rules) out of the consistency rules database (15..150 rules). The conditions of this experiment are expected to resemble the performance of this architecture with relatively large documents (like software engineering documents), when a modest amount of changes is made, and those changes affect a significant number of consistency rules.



1 critical consistency rule, 10 documents



10 critical consistency rules, 25 documents



20 checked consistency rules, 50 documents

Legend:

(1) 20 rules, total 40 rule elements to choose from, 1 changed element checked in each document

(2) 150 rules, total 3000 rule elements, 1 changed document element checked

(3) 20 rules, total 40 rule elements, 50 changed document elements checked

(4) 150 rules, total 3000 rule elements, 50 changed document elements checked

Fig. 3. Scalability graphs for performance of the agent-based architecture

The result of the performance evaluation provides an optimistic indication of linearity of performance degradation with the growth of number and complexity of consistency checks.

3. CONCLUSION

The consistency management architecture aims at co-ordinating the consistency functionality level from distributed agent interaction in exchange of information. Current research aims to enhance the existing consistency management framework by enabling coherent integration and co-operation in concurrent consistency checking across the network. The simulation data confirms that such an approach scales upward with the growth of complexity and number of the consistency checks. The test conditions are close to possible conditions arising in the domain of software engineering documents.

REFERENCES

1. T. Bray, J. Paoli, C. M. Sperberg-McQueen, Extensible Markup Language (XML), Recommendation, http://www.w3.org/TR/REC-xml, World Wide Web Consortium, 1998.
2. E. Clarence, J. Wainer, Groupware and Computer Supported Cooperative Work. In *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence*. Edt: G. Weiss, MIT Press, 1999.
3. S. DeRose, R. Jr. Daniel, E. Maler. XML Pointer Language (XPointer), Working Draft December 1999, http://www.w3.org/TR/WD-xptr, World Wide Web Consortium.
4. A. Finkelstein, E. Ellmer, W. Emmerich D. Smolko, A. Zisman. Consistency Management of Distributed Documents using XML and Related Technologies, UCL-CS Research Note 99/94 (Submitted for Publication), 1999.
5. D. Lange, M. Oshima, Programming and Deploying Java Mobile Agents with Aglets. Addison-Wesley, 1998.
6. U. Leonhardt, A. Finkelstein, J. Kramer, B. Nuseibeh, "Decentralised Process Enactment in a Multi-perspective Development Environment". In *Proc. 17th International Conference on Software Engineering (IEEE CS Press)*, 1995.
7. B. Nuseibeh. To Be and Not to Be. On Managing Inconsistency in Software Development. In *Proc. of 8th IEEE International Workshop on Software Specification and Design (IWSSD-8),* pages 164-169, March 1996.
8. O. Shehory, K. Sycara, S. Jha, Multi-Agent Coordination through Coalition Formation, In: *Intelligent Agents IV: Agent Theories, Architectures and Languages; 4th International Workshop ; proceedings / ATAL'97,* ed. M. P. Singh, A. Rao, M. J. Wooldridge, Springer, 1998.
9. D. Smolko, Using Mobile Agents to Address Consistency Issues in Distributed Document Management. In *Proc. of Doctorate Symposium, Automated Software Engineering '99*, October 1999.
10. G. Yamamoto, Y. Nakamura, Architecture and Performance Evaluation of a Massive Multi-Agent System. In *Proc. of the Third International Conference on Autonomous Agents,* pp. 319-326, (ACM Press), 1999.
11. A. Zisman, W. Emmerich, A. Finkelstein, Using XML to Specify Consistency Rules for Distributed Documents. Submitted for publication, 1999.