

# Supporting Distributed Cooperative Work in CAGIS

Heri Ramampiaro \*

Alf Inge Wang

Terje Brasethvik

June 14, 2000

## Abstract

This paper describes how the CAGIS environment can be used to manage work-processes, cooperative processes, and how to share and control information in a distributed, heterogeneous environment. We have used a conference organising process as a scenario and applied our CAGIS environment on this process. The CAGIS environment consists of three main parts: a document management system, a process management system, and a transaction management system.

**Keywords:** Web-based software engineering, Internet computing: JAVA, XML, Intelligent agent software, Database systems, Document modelling, Process modelling, Transaction modelling.

## 1 Introduction

After the introduction of the Internet, more and more projects are taking place in heterogeneous environments where both people, information and working processes are distributed. Work is often dynamic and cooperative and involves multiple actors with different kinds of needs. In these settings there is a need to help people coordinate their activities, share documents and information, and to manage the access to shared resources. While the web makes it fairly easy to distribute information, the web itself does not contain explicit mechanisms to plan and coordinate activities and tasks, organise, describe and classify information, or control access to - and ensure consistency of - project documents.

In the absence of "web-librarians" that can fulfil such tasks, the users themselves often have to figure out ad hoc solutions for doing this. To help the users, what is needed is a small set of powerful, easy-to-use and flexible tools that may be readily configured to support the task at hand. The CAGIS project - Cooperative Agents in the Global Information Space - aims to support such tasks by using a combination of software agents and small web-accessible tools.

This paper describes how our CAGIS environment, described in section 2, addresses the challenges or problems given above. In section 3, we present a conference scenario to make the problems and challenges more concrete, and the CAGIS environment is applied to this scenario in section 4. Section 5 discusses our approach and concludes the paper.

## 2 The CAGIS environment

The CAGIS environment consists of three main components: A system for handling of distributed documents and document understanding, a system for supporting cooperative processes in a distributed environment, and a flexible transaction management system for shared, distributed resources. The following three sub-sections will describe our effort in these research areas more in details.

### 2.1 Document models and tools

Documents published on the web have to be organised, classified and described to facilitate later retrieval and use. One of the most challenging tasks is the semantic classification - the representation of document contents. This is usually done using a mixture of text-analysis methods, a carefully defined (or controlled) vocabulary or ontology, as well as a scheme for applying this vocabulary when describing a document. The CAGIS document model toolset

---

\*Dept. of Computer and Information Science, Norwegian University of Science and Technology (NTNU), N-7491 Trondheim, Norway. Phone: +47 73 594485, Fax: +47 73 594466, Email: heri/alfw/brase@idi.ntnu.no

helps the users of a project group to do this semi-automatically, by way of a domain model expressed in a conceptual modelling language and by using text analysis tools as an interface to perform the actual classification and search. In other words, we use a conceptual model as a basis for creating meta-data descriptions (figure 1). These meta-data descriptions may then be accessed through our java-model viewer that enables search and browsing of documents through a standard web browser environment.

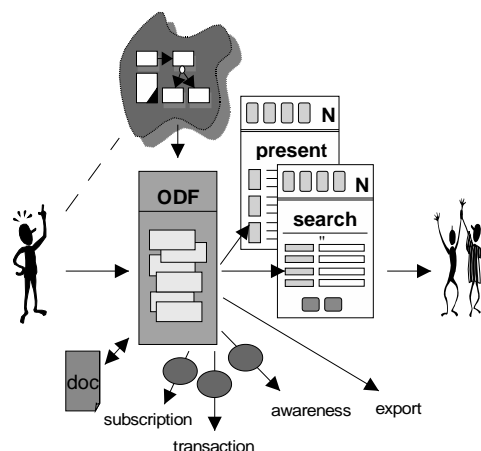


Figure 1: Conceptual modelling for meta-data descriptions

Fundamental to our approach is the use of a conceptual modelling language to define and visualise the domain specific vocabulary to be used in the classification and retrieval process. Conceptual modelling languages contain the formal basis that is necessary to define a proper ontology, yet at the same time they offer a visual representation that allows users to take part in the modelling, and to read and explore documents by interacting directly with the models. The conceptual modelling language may thus be used throughout the entire process of classifying and retrieving documents on the web. In our approach, we use the Referent model language [2] that is an ER-like language with strong abstraction mechanisms and sound formal basis.

Our approach may be described as a three-step process: *Domain Model Construction*, *Document Classification* and *Browsing & Retrieval* - outlined below.

**Domain Model Construction:** Conceptual modelling is mainly a manual process. However, our domain models must be somewhat related to the text of the documents to be classified, so we construct our model with some aid from a textual analysis tool. A reference set of documents from the domain is run through a word frequency analysis tool, which produces a list of high frequency terms from these documents. These terms are then taken as input candidates for the actual conceptual modelling task. This is a manual and cooperative task performed by a selected set of users. In order to prepare the finished domain model for later document classification, we then add lexical linguistic information to the model, i.e. the model is enhanced by adding a textual definition and a term-list for each of the concepts in the model. The term-list is a list of synonyms and conjugations for each concept that will be used when matching document text against model concepts.

**Document Classification:** Documents are classified by selecting domain model fragments that reflect the document content. This is performed semi-automatically by matching the document text against the term-lists for each concept in the model. Concepts found in the document are then shown to the user as a selection in a graphical model viewer and the user may manually refine the classification by selecting and deselecting concepts and relations. When the user is satisfied, the selected model fragment is translated into RDF-XML serialisation syntax (ref) and is stored as an "Object Descriptor File" (ODF) pointing to the document in question. The user also has to provide a selected set of properties for the document, such as its author, title etc. These attributes are also stored within the ODF.

**Browsing and Retrieval:** In order to retrieve documents, the users enter a natural language query phrase which is matched against the conceptual model in a similar way as in the classification process. The domain model concepts found in this search phrase (if any) are extracted and used to search the stored document descriptions.

Users may then refine their search by interacting with the model. Found documents are presented as list in a Web-browser interface by using the stored document attributes. We also have an enhanced "document reader", that is, when reading a document, all the terms in the document that matched a model concept is marked as a hyper-link pointing to the definition the model concept.

The layered architecture of our document tool is shown in figure 2. As mentioned, the main parts of the system are the web-enabled user interface and a set of servlets running on a standard web-server.

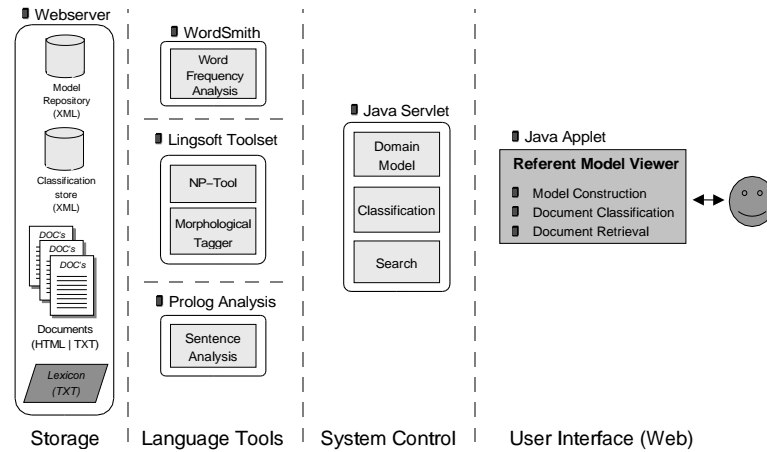


Figure 2: Overview of system architecture

- The user interface is centred around a Java-based Referent-model viewer. As mentioned, users may interact with the model, explore concept definitions and relations, and then use the viewer directly in order to perform both classification and retrieval.
- The Java servlets define the overall functionality of the system. They are invoked from the model viewer and coordinate the linguistic tools incorporated into the system.
- At an "intermediary" layer, between the servlets and the web-server, we use a number of linguistic tools analyse natural language phrases and give the necessary input to construct domain vocabularies and classify and retrieve documents. The Word frequency analyser from WordSmith is a commercially available application for counting word frequencies in documents and producing various statistical analyses. A Finnish company, Lingsoft, has two tools for analysing nominal phrases and tagging sentences needed for the classification and retrieval of documents. A smaller Prolog application for analysing relations between concepts in a sentence is being developed internally at the university. This application assumes a disambiguated tagged sentence and propose relations between the concepts on the basis of classification-specific rules. As an alternative to the tagger and the relation analyser, we are considering a parser linked to an extensive semantic lexicon.
- Finally, the documents and their classifications are stored at the web server in HTML/TXT and XML format. The domain model is also stored in XML and can be continuously maintained to reflect the vocabulary used in the documents being added to the document collection. The linguistic tools rest on lexical information that is partly stored in XML and partly integrated with the tools themselves.

A more detailed presentation of our approach and the system is given in [13, 14]

## 2.2 Process models and tools

A process centred environment (PCE) prototype has been developed to give process support to distributed, cooperative processes in CAGIS. The CAGIS PCE consists of three main components:

**Workflow System supporting Distributed Mobile Processes** This workflow system is used to model simple, repeatable workflow processes, and the system offers agenda-browser for the end-users. The workflow system allows an instanciated workflow model to be distributed as several process fragments on different workspaces. One

benefit of this is the possibility to adapt the workflow to local environmental conditions. The workflow instances are defined as XML-files located in the local workspaces, and can be changed any time. The ability to move workflow instances during enactment, can be used for reallocation of activities, dealing with exceptions (someone responsible for a particular activity is sick), and delegation of work. The Workflow system is implemented in Perl, providing a CGI-interface through a web-server. For a more detailed description, see [15, 16].

The Process Modelling Language (PML) for the workflow system defines a process as set of activities that can have pre-order relationships between them specified in XML syntax. An *activity* can specify a set of *pre-links* identifying what activities to be executed before, and *post-links* identifying activities to be executed after the activity current activity. The pre- and post-links can be written as URLs, and allow therefore the process to be distributed over several workspaces. Every activity definition specifies a code part. This code part is simply HTML, and can be used to simple present text, to specify a form, or to start a Java-applet. The term *process fragment* is used to name a group of activities in a workspace, which is one part of the whole process. A process fragment is specified by a name, a workspace (location), and a list of references to activities.

**Software Agents to support Dynamic, Cooperative Processes** While the workflow system described above takes care of simple, repeatable process, we use software agents to support more cooperative and dynamic processes. Software agents typically takes care of inter-workspace (inter-group) activities as negotiation activities (e.g., about of resource allocation), coordination of artifacts and workflow elements between workspaces, brainstorming, voting, marked support (in a multi-company scenario, we can perceive that agents act as buyer and sellers of services), etc. Our multi-agent architecture consists of four main elements:

- **Agents** An agent is set up to achieve a modest goal, characterised by autonomy, interaction, reactivity to environment, as well as pro-activeness. We have identified three main types of agents: (1) *Work agents* to assist in local production activities, (2) *Interaction agents* to assist with cooperative work between workspaces, and (3) *System agents* to give system support to other agents. Interaction agents are mobile, while system and work agents are stationary.
- **Workspaces** A workspace is a temporary container for relevant data in a suitable format to be accessed by tools, together with the processing (work) tools. It can be private, as well as shared. Files stored in a repository can be checked in and out to a workspace.
- **Agent Meeting Place (AMP)** AMPs are where agents meet and interact. AMPs provide agents support for doing efficient inter-agent communication. There can be different types of AMPs for different purposes. Each AMP will have a defined ontology, which the agents have to follow. We can perceive special AMPs for negotiation, coordination, information exchange, selling and buying services etc.
- **Repositories** Repositories can be global, local, or distributed, and are persistent storage of data. Experience Bases are one specific type of repository we can use in our multi-agent architecture to support community memory.

The multi-agent architecture is implemented in Java, using IBM Aglets framework to provide mobile agents, KQML is used for inter-agent communication, and ORBIX CORBA is used to offer communication to other applications and other agent systems. More detailed description of the multi-agent architecture can be found in [19, 11, 7].

**Agent-Workflow GlueServer** The Agent-Workflow GlueServer provides interaction between the workflow system and the multi-agent system. A **glue model** in XML defines the relationship between workflow elements and software agents. The **GlueServer** will offer services for a workflow activity to trigger an agent and vice versa. The GlueServer is implemented in Java, and ORBIX CORBA is used to facilitate communication with the agent system and workflow systems. More information about the GlueServer can be found in [18, 4].

Figure 3 shows a simplified illustration of how the different components in the CAGIS PCE interact. In figure 3, there are two workspaces, each running a workflow tool with a local workflow model. In reality, this workflow tool can be shared, and the local workflow models in the two different workspaces can have relationships between them. The figure illustrates two different ways that software agents can interact with workspaces. In the first way, the agents can interact directly with the user in the workspaces, using a graphical user interface to configure and interact with the agents. In the second way, the user does not interact with the software agents directly. All interaction with software agents goes through the GlueServer and the workflow tool. The workflow tool can

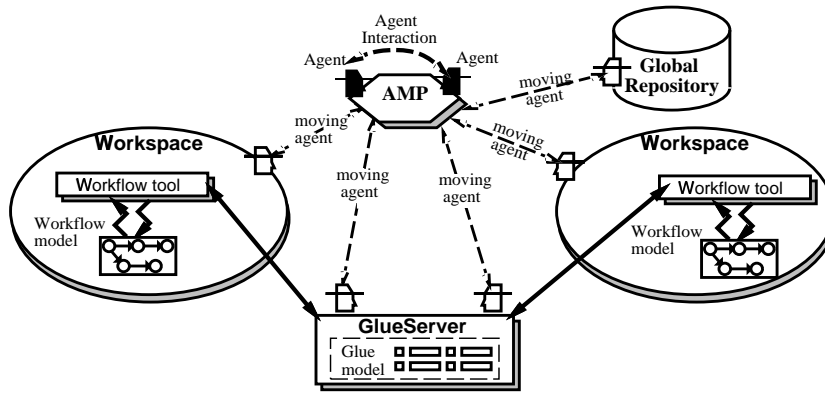


Figure 3: The CAGIS Process Centred Environment

activate an agent, or an agent can activate the workflow tool. The figure also shows that agents can be used to access repositories, but workspaces can also access files in the repository directly (not shown explicitly in the figure).

### 2.3 Transaction models and tools

A transaction is a basic work unit or a program segment executed to perform some function or task by accessing and manipulating a shared database. Transaction modelling concerns with capturing the essential characteristics of transactions. In general, these include transaction behaviour and applied constraints.

Transaction modelling in CAGIS was motivated based on the assumption that database systems will be used as resource management systems. The main purpose of the transaction system is then to control and manage access to a shared resource, and to make sure that this access is done according to prespecified consistency preservation constraints. This section briefly describes our effort in developing a transaction framework and a transaction management system for cooperating agents.

**The transaction specification framework** We have proposed a transaction framework to reason and specify customised, application specific transaction models [10].

The main purpose of this framework is to provide a configurable transaction models. This allows a user and application dependent customisation of transaction models. This is crucial to adjust the required degree of control to be provided through the transactions. Some situation may, for instance, require strict control for the correctness of data to be possible to preserve, other may see control as just a burden, and so on.

The framework is divided into two main parts: *Transaction characteristics specification* and *transaction execution specification*.

The characteristics specification is used to define the main properties of the actual transaction: ACID properties, relationship among the involved transactions, adopted correctness criteria (which is partly user-defined and partly prespecified), and applied policies (i.e. rules defining what concurrency control mechanism(s) to be used, and how and under which condition to use them). It is statically defined and must be done before the designated transactions are executed. The execution specification defines how the transaction execution is to be performed, in terms of composition of management operations (e.g., begin, commit, delegate and so on) and regular access operations (e.g., read, write, and so on). The execution specification must conform to the former, and has some fixed initial operations, while the remaining operations can be adjusted at run-time.

In this respect, the CAGIS transaction framework distinguishes between static and runtime dynamic specifications. Compared with related frameworks, such as ACTA [5], ASSET [3], and TSME [6], the main difference is on the dynamicity. Such a property is important since it is not always possible to predict all aspects of application in advance. This is particularly relevant when taking software agents as well as cooperative work into consideration.

A detailed presentation of this transaction framework is given in [10].

**The transaction management system** A system for the specification above is depicted in Figure 4. It is divided into two main components; a specification environment and a runtime management system.

- **The specification environment** provides a facility for a transaction model designer to specify the characteristics of transactions as needed. The same environment also allows a user to specify a set of operations

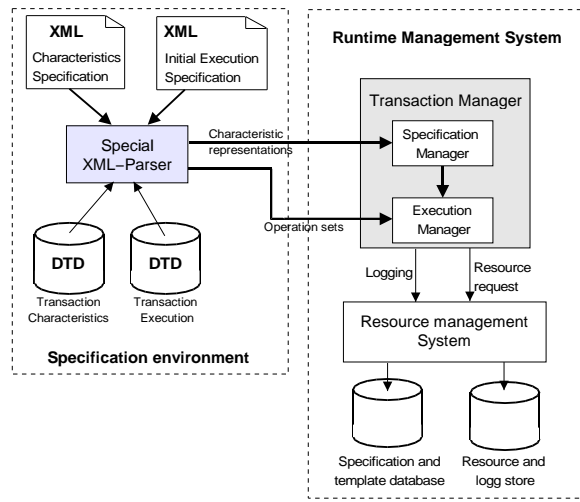


Figure 4: Transaction management architecture

to be run by a transaction. Both of the aforementioned specifications are done in XML. A special XML parser is used to validate specifications, against the prespecified DTD. This parser transforms (1) the specification of characteristics into an internal representation of transaction characteristics, and (2) the execution specification into a set of operations. They both are used by the transaction manager to control and monitor transaction executions. Further the transaction specifications are kept in a specification database. To avoid redundant specifications, the specification environment provides a browsing facility to allow the designer to check, if the transaction characteristics are already in the specification database. This implies that future adjustments can be performed without needing to do the specification from scratch. Finally, if changes are made, the designer is asked whether the current specification is to be saved as a new version or to replace the old one.

- **The runtime management system** consists of a transaction manager and a resource management system. *The transaction manager* is responsible of managing the specification and execution of transactions, and to ensure that any transaction execution is done according to the specified characteristics. For example, if an ACID model is defined, it will enforce atomic and isolated execution. Correspondingly, if property atomicity is relaxed, then the same manager will ensure that any failure would not necessarily cause global rollback. Instead, partial abort can be issued. Now, as shown in Figure 4, the transaction manager again consists of a specification manager and an execution manager. *The specification manager* is responsible of controlling all necessary representations from the specification environment are complete. In other words, it has to make sure that the specification of transaction characteristics can be supported and that all necessary semantics are represented. If such a specification is not fully satisfiable it will either notify the designer and ask him/her to adjust the specification, or it will choose a closest supported specification that can be found in the specification database. Otherwise, it makes the characteristics information available to the execution manager. Then, this manager is now responsible of ensuring that the transactions are executed consistently with respect to the transaction characteristics and the execution information. Based on the specification, the execution manager issues the necessary and suitable transaction management operations. This means, that it issues *begin*, *abort* or *commit*, and other management operations that the user has specified.

*The resource management system* is responsible of managing and providing system resources for running transactions. It also maintains the execution information of the running transactions and uses this to handle transaction aborts and system recovery. Finally, the resource management system is responsible of making sure that committed results are kept in a persistent store.

The transaction management system described above was implemented in a working prototype [12, 8] based on Java and the IBM Aglet-workbench. It has served as a test-bed for the transaction specification framework.

### 3 Conference scenario

This section describes briefly a conference organising process we are using as a scenario, previously presented in [9]. The seven main activities of the conference organising process are shown in bold-face in next paragraph.

First the Program Chair will initialize the process by **Planning and announcing the conference**. People wanting to attend to the conference will submit their papers, and PC members will **Record submitted papers** as well as information about the authors. Then, **Reviewers will be chosen** based on their expertise, and the **Paper reviewing** starts. The PC members will then **Collect reviewing results**, and a review meeting will be held to **Determine acceptance of papers**. Accepted papers will then be **Grouped into sessions** and a final program including a time-table for conference sessions will be produced.

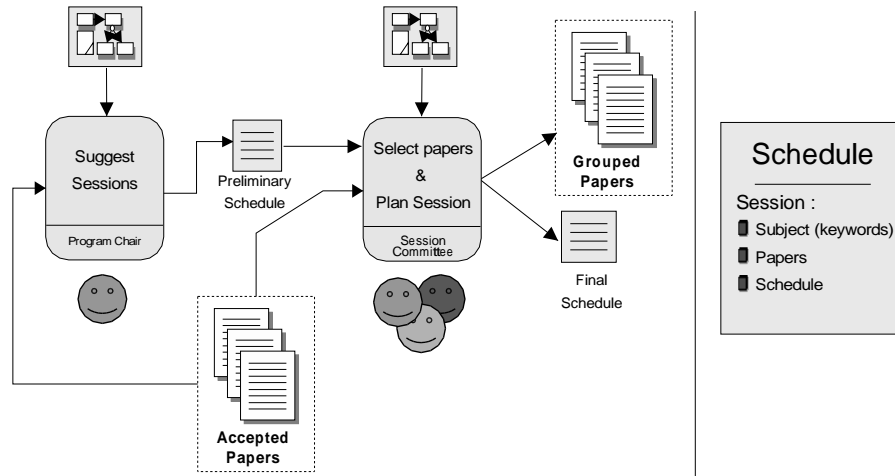


Figure 5: Group Accepted Papers into Sessions

In this paper, we will focus on the last main activity *Group Accepted Papers into Sessions*. Figure 5 shows the two main sub-activities of this activity.

**Suggest Sessions** Program Chair is responsible for this activity, and it can be decomposed into these process steps:

1. Match all papers against a document model defining terms and expressions, and the relationships between them for the research domain.
2. Suggest session division according to subjects
3. Create a preliminary session schedule
4. Set Up Session Committees (from Program committee members)

**Select papers & Plan Sessions** All members of Sessions Committees are responsible for this activity, and it can be decomposed into these process steps:

1. Determine session subject & goals: An *initial* Session description will contain session subject and goals.
2. Check papers for session: Sessions Committee members should mark papers relevant for a session to notify their interest. Papers will be marked "*possible*".
3. Paper allocation: If papers are marked by several Session Committees, Session Committees must negotiate about who is going to get the paper. Papers allocated to a session will be marked "*taken*".
4. Check timeslot for session: Each session committee will mark the timeslot for the session.
5. Session allocation: Sessions that have the same timeslot will negotiate. When all sessions are allocated, the result will be added to the session description. The session description will now have the state "*final*".
6. Publish session description: Each Session Committee will publish their session description to the other Session Committees and Program Chair.

In this paper we assume that the Program committee members will be distributed on different locations and the work with organising the conference will be done only through computer interaction (without any physical meetings).

## 4 The CAGIS environment applied on the scenario

This section outlines how the CAGIS environment, consisting of tools and models to support documents, processes and transactions, can be applied on the scenario described in section 3. Our suggested architecture to support this scenario is shown in figure 6.

The two main activities we are focusing on, **Suggest Session** and **Select papers & Plan Session**, are executed by the Program Chair and the Sessions Committees respectively. In our solution we have therefor chosen to model the scenario using one workspace for Program Chair and one workspace for each Session Committees. Each workspace has a local process defined in a process model, and a workflow tool that enacts this process model. The process models are defined according to the process steps defined for **Suggest Sessions** and **Select papers & Plan Session** as given in section 3.

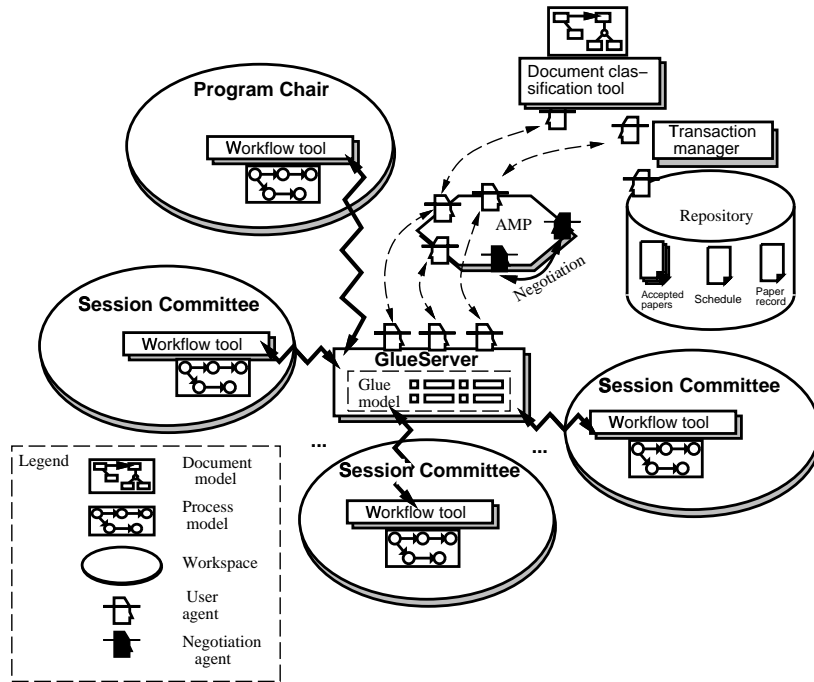


Figure 6: The CAGIS framework applied on the scenario

The Program Chair's first process step is to divide all papers according to their topic. This is done by using the **Document classification tool** to match all papers against the domain model. The domain model defines the vocabulary of keywords, extracted from the preliminary conference topics and from the submitted papers. The matching of papers against the domain model is visualised in the document model viewer, and the Program Chair may interact with the model viewer to achieve the proper subject division of papers.

The **Workflow tool** will here notify the **GlueServer** that will initialise a *document agent* that may be used to access the documents through the document servlet. In the two next process steps, the workflow tool will present Program Chair with necessary documents and tools for creating a preliminary session schedule and setting up session committees. When session committees are selected among PC members, the session committee members will be notified through email describing what session committee to attend and what workspace to access.

The Session Committees will then start to work in their workspaces according to the process model enacted by the workflow tool (remember that all conference organising work will be done distributed on computers). First they have to determine session subject & goals, the workflow tool will notify the GlueServer that will initialise *brainstorming agents* for each session committee member. The result of this brainstorming process will end as an initial session description written by session chair. The next step of the process will be for the session committees to choose what papers to be in the session. Here the workflow tool notifies the GlueServer to initialise *paper select agents*. The paper select agents will retrieve information about available papers, and let the session committees mark interest of papers. The paper select agents will then mark papers in the *Paper record* in the repository (see figure 5). The result from marking papers will be returned to the GlueServer. If papers have been marked by several session committees, negotiation agents will be initiated to negotiate about what session is going to get the paper. If the negotiation process goes into a deadlock, Program Chair will be notified, and he/she will make a final

decision. The next step, is for the session committee to mark a timeslot for the session. This process works exactly like paper selection, but *session selection agents* will be used instead. When all session committees have selected their timeslots, the final session description is published to all, and the final conference program can be produced.

The **transaction manager** is responsible of managing the integrity of the document in the repository, and to ensure that agents accomplishing their task always leaves the system in a consistent state. Based on the description above, the session committees share both the schedule document and the paper record. Therefore, conflicts are likely to occur. There are several possibilities to solve this problem. For example, one may provide an exclusive lock for each access, thus prohibiting other to see any changes until the process is finished. This may however be unacceptable since it might delay the session arrangement process. An other possibility is that instead of hindering other session committees from total access, read permission is provided read. This allows other committees to see the intermediate changes, and therefore eases their decision process. An other scenario again is to permit simultaneous updates (i.e., write/write conflict). However, achieving consistency is possible only if the system supports multiple-version handling, with a sophisticated merging mechanism to capture all possible changes.

Next, tasks to support the session selection process are assigned to agents. As mentioned, this may involve document access too. To allow the transaction management system to ensure consistency, all agent method invocations involving repository access are managed as part of transaction execution. This also ensures that conflicting document access is managed properly. Moreover, suppose that a transaction consisting of several agent method invocations is initiated by the GlueServer. Then, assume that one of the involving agents fails, for example, while selecting papers from the paper record. Using traditional ACID transactions, this would cause a global rollback, thus discarding all changes made so far, and killing all associated agents. However, if a lot of efforts have been invested, then to start all tasks from scratch may be expensive. To cope with this, we model each agent method invocation as a subtransaction of that executed by the GlueServer. Therefore, instead of aborting the transaction and killing all involved agents, the transaction manager allows the failing agent to just undo some of its changes. Other agents may proceed as normal.

## 5 Conclusion

This paper has presented the CAGIS toolset and its applications to a conference organisation scenario. The CAGIS toolset consists of a set of separate tools that may be used together to provide support for cooperative work across the web. The three major components of CAGIS are the Workflow tool, the Document classification tool and the Transaction manager. Each of these tools is implemented in true Web style, i.e. they are built around a standard Web server and use XML as a data storage and interchange format. These tools may all be configured according to the actual situation and use. The *Workflow tool* allows for the creation of individual workspaces to support the activities of the workflow, in addition, the workflow tool offers the ability to enact the part of the process model that the workspace supports. The *document classification tool* uses a domain specific vocabulary, the domain model, created by the users to classify and search for documents. The *transaction tool* offers support for the specification and execution of customised and application-specific transaction models. The transaction manager thus offers the ability to design the required correctness criteria and to define and execute transactions that enforce these criterias on shared resources. Central to our system, and binding the individual tools together, is a GlueServer. The *GlueServer*, configures a set of software agents that can activate the different CAGIS tools. The *glue model* defines the relations between the individual workflow elements that may reside in different workspaces and the software agents that may be used to access the individual tools. This way, the various components of the CAGIS toolset may be used together in order to provide situation specific cooperative support.

Our CAGIS environment is not only applicable to conference organisation processes. The CAGIS environment can be used to support any process where people are working together, and where people and information are distributed. Examples of such processes can be cooperative software engineering processes, distributed educational processes, distributed organising processes, processes of selling and buying merchandises on the web etc. All these processes are characterised with distribution of people and information, and require people to interact and cooperate to reach the goal of the process.

Furthermore, when combining the tools in the CAGIS environment, we enhance the functionality of one specific CAGIS tool. In [1], an example of how the document models and tools can enhance the CAGIS multi-agent architecture is given. The document models and tools are here used to model the agent ontology, which define the language software agents can speak. In [17], the transaction models and tools (in this paper called workspace manager) offer a way managing consistency of changing workflow models. This means that the CAGIS environment offers a selection of tools, which can be used in different combinations to give specific support. Future work will investigate more thoroughly what tools to pick for different scenarios.

## References

- [1] Alf Inge Wang and Anders Aas Hanssen and Bård Smidsrød Nymoen. Design Principles for a Mobile, Multi-Agent Architecture for Cooperative Software Engineering. Submitted to Software Engineering and Applications'2000 (SEA'2000).
- [2] Arne Sølvberg. Data and what they refer to. In P. Chen, editor, *Conceptual modeling: Historical perspectives and future trends*, Los Angeles, California, USA, 1998. 16th Int. Conf. on Conceptual modeling.
- [3] Alexandros Biliris, Shaul Dar, Narain H. Gehani, H. V. Jagadish, and Krithi Ramamritham. Asset: A system for supporting extended transactions. In Richard T. Snodgrass and Marianne Winslett, editors, *ACM SIGMOD International Conference on Management of Data (SIGMOD 94)*. ACM Press, May 1994.
- [4] Bjørn Haakenstad. GlueServer, support for integrating workflow-systems with interactive agents. Technical report, Norwegian University of Science and Technology (NTNU), March 2000. Technical Report, Dept. of Computer and Information Science.
- [5] Panos K. Chrysanthis and Krithi Ramamritham. Synthesis of extended transaction models using ACTA. *ACM Transactions on Database Systems*, 19(3):450–491, Sept. 1994.
- [6] Dimitrios Georgakopoulos, Mark F. Hornick, and Frank Manola. Customizing transaction models and mechanisms in a programmable environment supporting reliable workflow automation. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):630–649, August 1996.
- [7] Anders Aas Hanssen and Bård Smidsrød Nymoen. DIAS II - Distributed Intelligent Agent System II. Technical report, Norwegian University of Science and Technology (NTNU), January 2000. Technical Report, Dept. of Computer and Information Science.
- [8] Lars Killingdal and Mufriid Krilic. Programmerbare transaksjoner – prosjektoppgave. Technical report, Norwegian University of Science and Technology, April 2000. Student Project.
- [9] T. W. Olle, H.G. Sol, and A. A. Verrijn-Stuart. Information Systems Design Methodologies: A Comparative Review. North-Holland, 1982.
- [10] Heri Ramampiaro and Mads Nyård. Cagistrans: A transaction framework to support cooperating agents. Technical Report IDI-5/00, Norwegian University of Science and Technology, 2000.
- [11] Geir Prestegård, Anders Aas Hanssen, Snorre Brandstadmoen, and Bård Smidsrød Nymoen. DIAS - Distributed Intelligent Agent System. Technical report, Norwegian University of Science and Technology (NTNU), April 1999. Technical Report, Dept. of Computer and Information Science, 387 p.
- [12] Rune Selvåg. Web-transactions. Master's thesis, Norwegian University of Science and Technology, 2000.
- [13] Terje Brasethvik and John Atle Gulla. Semantically accessing documents using conceptual model descriptions. In P. Chen, D.W. Embley, and S.W. Little, editors, *Advances in conceptual modeling*, Paris, France, November 1999. WEBBCM\*99.
- [14] Terje Brasethvik and John Atle Gulla. Natural language analysis for semantic document modeling. In E. Metais, editor, *Proceedings on 5th International Conference on Application of Nature Language to Information Systems (NLDB'2000)*, Versailles, France, June 2000.
- [15] Alf Inge Wang. Experience paper: Using XML to implement a workflow tool. In *3rd Annual IASTED International Conference Software Engineering and Applications*, Scottsdale, Arizona, USA, 6-8 October 1999.
- [16] Alf Inge Wang. Support for Mobile Software Processes in CAGIS. In *Seventh European Workshop on Software Process Technology*, Kaprun near Salzburg, Austria, 22-25 February 2000.
- [17] Alf Inge Wang. Using Software Agents to Support Evolution of Distributed Workflow Models. In *In Proc. International ICSC Symposium on Interactive and Collaborative Computing (ICC'2000)*, page 7, Wollongong (near Sydney), Australia, December 12-15 2000.
- [18] Alf Inge Wang, Reidar Conradi, and Chunnian Liu. Integrating Workflow with Interacting Agents to support Cooperative Software Engineering. Submitted to Software Engineering and Applications'2000 (SEA'2000).
- [19] Alf Inge Wang, Chunnian Liu, and Reidar Conradi. A Multi-Agent Architecture for Cooperative Software Engineering. In *Proc. of The Eleventh International Conference on Software Engineering and Knowledge Engineering (SEKE'99)*, pages 1–22, Kaiserslautern, Germany, 17-19 June 1999.