

# Cooperation Services in the Construct Structural Computing Environment

Uffe K. Wiil, Samir Tata and David L. Hicks

Department of Computer Science and Engineering  
Aalborg University Esbjerg, Niels Bohrs Vej 8, 6700 Esbjerg, Denmark  
{ukwiil,tata,hicks}@cs.aue.auc.dk

**Abstract.** Environments for structural computing have seen significant recent development. Generally, they provide different hypermedia structuring facilities to standard desktop applications. Users within these environments may perform activities involving several standard desktop applications and handle several types of hypermedia structures. In this working context, users need cooperation facilities that help them coordinate their activities and manage their roles in cooperation. This paper describes the design, development, and integration of a range of cooperation services into the Construct structural computing environment.

## 1. Introduction

In the last few years, cooperative applications have seen significant development and a real improvement of their underlying technologies. Among others, the particular class of cooperative applications dedicated to support project-based organizations is actually emerging. Our work is concerned with this particular class of cooperative applications, which aggregate several partners around a common project. Cooperation in these applications occurs in the more common case through sharing of objects: the products of the project. Shared objects in developed cooperation environments are considered as atomic without structure and isolated without dependencies.

Structural computing research [23,27,28] (which originates from open hypermedia research [38]) aims at the development of environments that provide different types of structure services (e.g., navigational [3], argumentation [29], taxonomic [26], spatial [19], etc.) that can be used to structure different types of documents such as the shared objects in project-based organizations. Due to the wide spread success of the World Wide

Web [2], the most well-known type of structuring domain is the navigational, which allows different shared objects to be linked together.

In many instances, tools and frameworks for cooperative applications have been developed without real support for hypermedia structuring facilities (e.g., metadata management). Moreover, environments for structural computing lack facilities to support cooperation. We think that it is important to combine the computer-supported cooperative work (CSCW) and structural computing fields [16]. The potential synergism they hold for each other can be very effective in supporting projects involving several cooperating partners and handling structural data.

The combination of the CSCW and structural computing fields can be seen from several points of view. One can develop research in the CSCW field by using results from the structural computing field and support cooperation on hypermedia documents. In this context, Haake and Wilson propose an approach based on the technologies of hypertext systems, computer networks, and database management systems to support collaborative writing of hyperdocuments in SEPIA [14]. One can also develop the CSCW domain as an application of the structural computing field. For example, Haake proposes in [13] to model coordination tasks with a structural computing approach. Finally, one can develop the hypermedia field by using results from the CSCW field and support cooperation in open hypermedia systems [12].

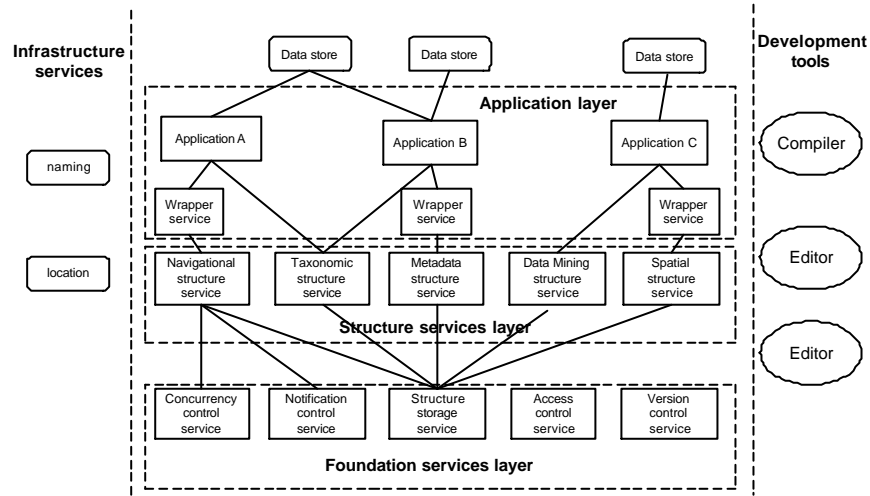
The present work takes place in the latter context. We aim to combine research results from the CSCW and structural computing fields by supporting cooperation services in a structural computing environment and developing a proof-of-concept implementation of these services in the Construct structural computing environment (or simply Construct) [35].

The paper is organized in the following way. Section 2 provides a brief overview of Construct. In Section 3, we describe the design of a set of cooperation services. We focus on services that support mainly asynchronous and informal cooperation in a structural computing environment. Section 4 describes the implementation of the proposed cooperation services in Construct. In Section 5, we touch upon present and future work relating to cooperation facilities in Construct (i.e., services for formal cooperation). Related research is examined in Section 6. Section 7 concludes the paper.

## 2. The Construct Structural Computing Environment

Existing structural computing environments take the main ideas from open hypermedia research (separation of data and structure and provision of linking services to existing desktop applications) a step further and introduce multiple hypermedia domains within a single environment [24]. Construct is a structural computing environment [5] based on previous open hypermedia research efforts that focused on the provision of infrastructure services and development tools (e.g., DHM [11], HOSS [24,25] and HyperDisco [39,40]).

A recent trend in both open hypermedia and structural computing environments is towards more open and modular environments in which structure services and other services are developed as separate components with well-defined interfaces [37]. Following this trend, Construct services are decomposed into components that can be used independently of one another [35]. Figure 1 presents the conceptual architecture of Construct.



**Figure 1:** Conceptual overview of the Construct structural computing environment.

Construct provides three basic architectural layers: application layer, structure services layer, and foundation services layer. Services belong to a layer depending on the type of service they provide. The environment is designed in order to allow each layer to be open to new services. The environment hosts different categories of software components:

applications, wrapper services, structure services, foundation services, infrastructure services, and development tools.

The application category includes desktop applications (e.g., Netscape, Microsoft Word, and Emacs). These applications are integrated (modified, extended, or wrapped) in the Construct environment either directly or through a wrapper service. A wrapper service is a service that helps integrate a particular desktop application by mediating the communication between a structure service (e.g., a navigational service) and the application. Integrated applications can make use of different types of structure services to organize data located in data stores (e.g., file systems, web servers, CD-ROM's, etc.). This makes applications (with the help of wrapper services) responsible for merging the data (located in data stores) with the structure (located in structure stores in Construct) at runtime. When an application has loaded a document from a data store, it queries Construct for structures belonging to the document. The structures (i.e., links) are then shown (superimposed) in the document at the appropriate places. The application (and/or wrapper) is also responsible for providing a user interface to manipulate (create, update, and delete) the structure.

Each type of structure service provides a different set of structural abstractions (e.g., navigational, metadata, taxonomic, and argumentation) that support different application domains. The structure services depend on the very basic services provided by the foundation services layer. Example foundation services include structure storage services, concurrency control services, notification control services, access control services, and version control services. Infrastructure services provide essential services that enable the other services to operate in a massively distributed environment. Typical infrastructure services are naming and location services. Finally, Construct provides different development tools to assist service developers in constructing new services. Existing development tools include the Construct Service Compiler [36], UML Tool [34], and a specialized version of the Emacs editing tool [34,35]. These development tools can generate component skeletons based on high-level service specifications in UML or IDL. Additional details about Construct can be found in [5,33-37].

### 3. Cooperation Services Design

An important application area to which Construct can be applied is the support of CSCW applications. Construct's approach to provide infrastructure services is well suited to the provision of facilities to support cooperation. In addition, Construct supports the integration of information across several applications and hypermedia domain boundaries, which constitutes a good foundation for data sharing and cooperation.

To help guide the requirements and design of new cooperation services in Construct, we analyze a scenario from the domain of architectural design and building construction. This case study originates from a research project involving France Télécom, the ECOO<sup>1</sup> research team, and the CRAI<sup>2</sup> research team.

The scenario is presented in Section 3.1 and analyzed in Section 3.2. The following Sections (3.3 and 3.4) present the design of new services based on requirements from the scenario.

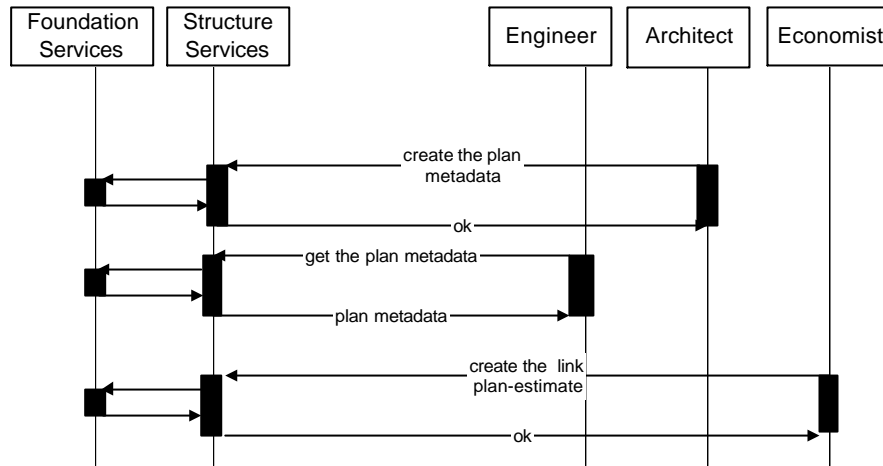
#### 3.1 Scenario

Assume that in a given architectural design project there are three cooperating partners: an architect, an engineer, and an economist. The architect is the person responsible for the production of the final plan of the building and all of the related documents. However, the engineer can be in charge of some parts of this plan or of some technical details. All along the design process, the economist produces cost estimations corresponding to the current version of the plan. The architect reads these cost estimations in order to check the compatibility of the project with the budget conditions.

---

<sup>1</sup> Environments for COOperation – <http://www.loria.fr/equipes/ecoo/english/>

<sup>2</sup> Research Center in Architecture and Engineering (Centre de Recherche en Architecture et Ingénierie) – <http://www.crai.archi.fr/>



**Figure 2:** A cooperative interaction scenario using a UML interaction diagram.

In Figure 2, we present a small example of a cooperative interaction scenario (using a UML interaction diagram) involving the architect, the engineer, and the economist working with Construct tools and services. Only the actions involving operations on structure are shown in the diagram. All other operations such as creating, opening, and updating documents residing outside Construct (in some data stores) are not shown in the diagram.

*First, the architect opens and updates the plan using his favorite application (not shown in Figure 2). After that, he creates metadata using the Construct metadata service for the current plan that will later be used by the engineer to perform her work. This metadata may represent some rules (specifying some technique choices or the sketch of the plan) established by the architect (the person responsible for the plan design) that will be followed by the engineer in her work. Later, during the process of the plan design, the economist needs to produce estimates of the building construction cost corresponding to the current plan. For this reason, she creates a new estimate using her favorite application (not shown in the Figure 2). Then, she creates a link between the new estimate and the current plan. She uses the Construct navigational structure service for this.*

### 3.2 Analyzing the Scenario

By analyzing this scenario, we have found two primary areas in which new services must be added to Construct to support cooperation:

**Awareness Service.** When the architect or the engineer changes the plan, the economist has to (must) be made aware about this modification in order to update her estimate. Thus, Construct needs an awareness service, which allows the different cooperating partners to be aware of the current status and the activity of others. This awareness service will act as an integrating mechanism prompting cooperating partners to talk to each other, discuss the plan, show results, and update them.

**Session Service.** The architect and the economist on one hand and the architect and the engineer on the other hand constitute two cooperating groups with different relationships and goals. Thus, Construct needs a session service, which allows the separation of the cooperation into two cooperation types in two cooperation contexts (i.e., two sessions). The session service will create and manage sessions and allow cooperating partners to start, stop, join, leave, and browse sessions.

In the following, we present the design of the session and awareness services that we call cooperation services and also discuss the implications of these new services upon the Construct naming service.

### 3.3 Naming Service Implications

To support the type of cooperation examined in this paper, we have found the need of services to manage cooperative sessions and group awareness. These two services place some specific requirements on the Construct naming service, which must be able to associate Construct components with names. The naming facilities are necessary especially in cooperation situations, which can involve several applications and consequently several services. When, for instance, a new service is added to the system, the service can subscribe to the naming service and then it can be used by applications without the need for hard coding naming information into the environment.

The Construct naming service allows services to locate other Construct components by a user definable name. The system allows associating an object (i.e., a user, a group, a structural service, and a foundation service) with a name. Once the name is associated with the object, any Construct service or application can access the object through the name.

The naming service is both very simple and very useful. It maps Construct object references to abstract names. New services can be registered with the naming service and so can be used by Construct services and applications. The IDL specification of the naming service is given below.

```
interface Naming{
    void register(in Namingable object);
    void remove(in String name);
    Namingable lookup(in String name);
};
interface Namingable{
    String getName();
};
```

### 3.4 Session Service Design

A session is a context allowing cooperative users to start, stop, join, leave, and browse collaborative situations across the network [17]. Within a session, users can connect from various locations to the structural computing environment in order to work together on shared artifacts such as a whiteboard and use multimedia conferencing tools to communicate. The context of a text document being edited within a shared editor and a videoconference among a number of participants are examples of sessions.

In order to create sessions and allow users to work within sessions, Construct needs a service to manage sessions. A session management service determines the way in which the users of a cooperative application join together into a collaborative endeavor [7]. The major functions of this service are the session information management and the command management. In the following, we describe broadly these facilities and we give their IDL specifications. After that, we describe the interaction of the session service with other Construct services.



### 3.4.1 Session information management

The session information management function creates sessions and updates information within active sessions. A simplified IDL specification of this facility is given below. Among other things, the session information management maintains the list of online users working in a given session: adds/removes users to/from a given session

```
interface SessionInformationManager{
    attribute String name;
    attribute Vector onlineUsers;
    string getName();
    void setName(in String newname);
    void addOnlineUser(in String name);
    void removeOnlineUser(in String name);
    Vector getOnlineUsers(in String group);
    Vector getAllOnlineUsers();
    void close();
};
```

### 3.4.2 Command management

The command management function opens and closes sessions. It allows users to log in (join) and log out (leave) and provides logged in users with facilities to execute commands on shared data using any structural service in Construct. The IDL specification of this facility is given below.

```
interface CommandManager{
    string create();
    void close(in String name);
    void join(in String name, in String password, in String
        group, in String session);
    void leave(in String name, in String session);
    void handle(in Command cmd, in String session);
};
```

### 3.4.3 Interaction with other Construct services

The session management service depends on other Construct services in the following ways.

**Foundation services:** Like almost all Construct services, the session management services depend on the store service. The store service is used

to save objects representing session information (session names, online users, etc.). In addition, the session service may use the store service to keep user preferences, personal data, application defaults, etc. In order to control access to the sessions, the session management service uses the access control service. In fact, it controls whenever a user has the permission to create or join sessions, to issue commands or to get the online users in a given session.

**Naming service:** The session management service depends on the naming service. The latter is used to select users and groups and to select and to locate Construct services (that the session service may be used to execute users' commands).

**Other Construct services:** To allow users to work together on shared objects using several Construct services, the session management service can interact with all structure services. It sends commands that will be handled by these services.

### 3.5 Awareness Service Design

In cooperative systems, users can be distributed in space, in time, and even over organizations. To interact effectively they need a mechanism that allows them to be aware of others: who is doing what? Coworkers need to be aware of the current status and the activity of others. For them, the awareness acts as an "integrating framework" by allowing them to talk to each other, discuss cooperative artifacts, show results, and update them. The need of supporting awareness in cooperative applications has been identified in several works. Some applications that support awareness include Quilt [10] and GROVE [9] that are summarized in [6].

The term awareness has a common usage to indicate mindfulness, or being conscious of surroundings, situations, or other people. Since the literature has not set a technical definition of the term awareness, we can, like in [8], present a definition based on separating the concept into two components, which have been investigated in the literature:

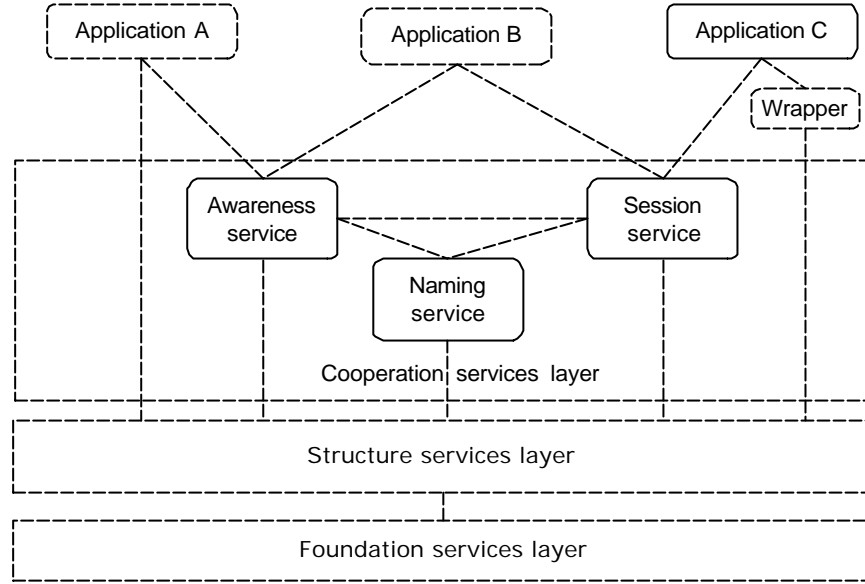
- ?Presence and location of coworkers;
- ?Previous and current tasks.

To detect the presence and location of coworkers and determine the previous and the current tasks, we propose a new awareness service in Construct based on the naming, session, and notification services. For instance, using the naming service and the operation *getOnlineUsers* of the session service the Construct awareness service can locate the online users. In addition using the Construct notification service, the awareness service can get the posted messages reflecting the activity of coworkers. The IDL specification of the awareness service is given below (the concept of events is described in Section 4.1).

```
interface AwarenessManager {
    attribute java::util::Hashtable users;
    attribute java::util::Vector messages;
    attribute java::util::Vector events;
    void postEvent(in Event event);
    void getEvents(out java::util::Vector events);
    void getFirstEvent(out Event event);
    void grantAllServicesAwareness(in String user);
    void grantServiceAwareness(in String user, in String
service);
    void getMessages(out java::util::Vector messages);
    void getFirstMessage(out Message message);
    void addMessage(in Message message);
    void getUsers(out java::util::Hashtable users);
    void addUser(in User user);
};
```

#### 4. Construct Prototype Implementation

This section presents a prototype implementation of the cooperation services as designed in Section 3. Currently, the new cooperation services are placed in a separate layer in the architecture – the cooperation services layer (see Figure 3). Section 5.1 contains a discussion of the advantages and drawbacks of placement of the cooperation services in different layers in the Construct architecture.



**Figure 3:** A new cooperation services layer added to Construct.

Section 4.1 provides a description of the new cooperation services. In Section 4.2, we revisit the scenario from Section 3.1 and describe how the collaborative activities it involves can be supported with the new cooperation services together with existing Construct services.

#### 4.1 Construct Cooperation Services

Existing Construct structure services such as the navigation [33] and metadata management [21] services make use of the store foundation service. To support informal cooperation, we implemented the awareness service, the session service, and a log foundation service.

The log foundation service is used to log events and operations performed by applications using structure services. It is designed as a database table. Each entry in this table stores information about an event (e.g., the user  $u$  has joined session  $s$ ) or an operation (e.g., the user  $u$  has created metadata for the data item  $d$ ). The information includes the sender (the user or the application), the level of importance (not important, normal, important, very important), a message describing the event, the date, the time, and the session within which the event was produced. If the user is not working in

cooperation with other users, many of these attributes are assigned a default value.

The awareness service uses the log foundation service to get events produced by cooperating users. It filters events and sends them to the users interested in them. The filters are based on:

- Level of importance: a user might work in an isolated manner and want to receive only very important events;
- Users: a user may want to receive only events generated by specific users;
- Services: a user may want to receive only events produced by users using some specific Construct structure services;
- Sessions: a user may want to receive only events produced by users working within some specific sessions; etc.

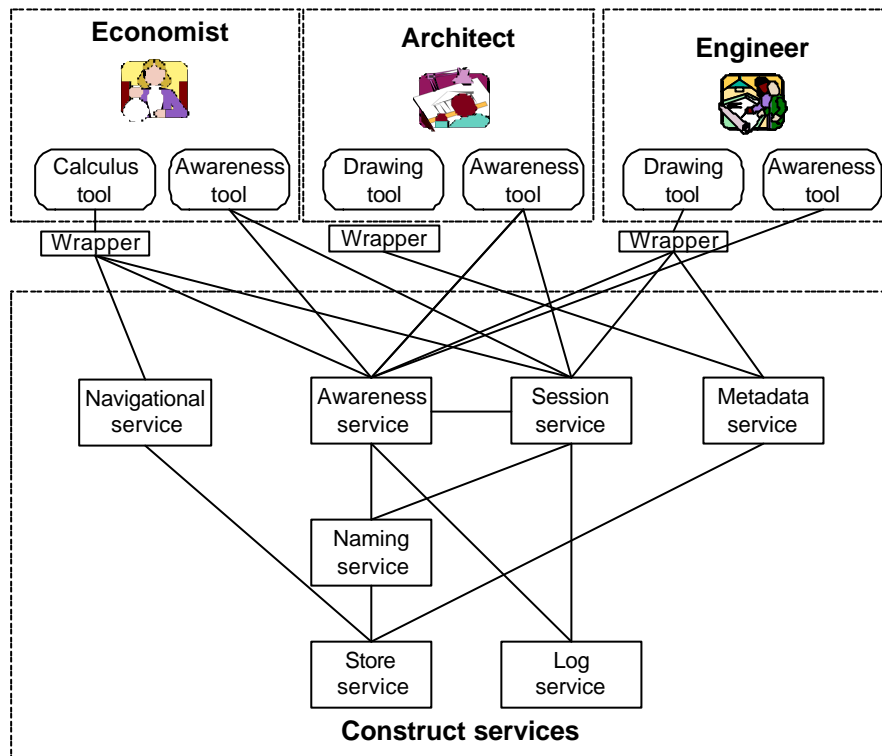
After filtering events, the Construct awareness service displays them in the awareness tools of the cooperating users. Additional ideas for future work relating to filtering structure events are presented in Section 5.3.

We developed an awareness tool that enables cooperating users to interact with the collaboration services and that displays collaboration related information. This approach allows the use of collaboration services without requiring the modification of user applications. Another solution would be to integrate the display of collaboration related information into user applications. However, this solution would require the modification of all applications being used during collaboration sessions.

The user interface of the awareness tool was deliberately not given high priority. The focus of the implementation work was instead placed on developing a running prototype of the tool. Section 5.4 describes plans for future work that will give the user interface of the awareness tool its appropriate attention.

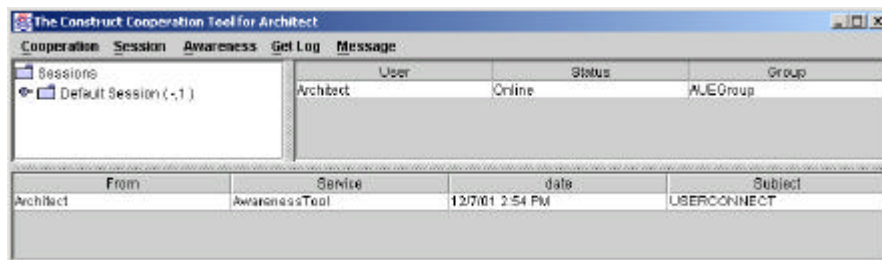
## 4.2 Scenario Revisited

In the following, we present a description of how the scenario from Section 3.1 can be supported within the modified Construct environment. The tools and services used in the revisited scenario are depicted in Figure 4.



**Figure 4:** The tools and services used in the scenario.

The Construct awareness tool (Figure 5) is divided into three main parts (see the screen shot below): the session part (upper left), the user part (upper right), and the event part (lower).



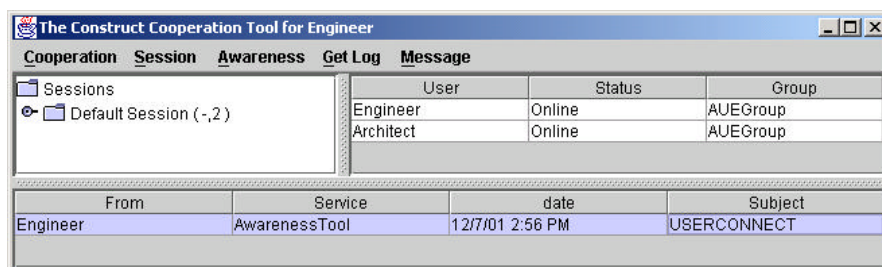
**Figure 5:** The user interface of the Construct awareness tool.

The session part lists the created sessions. Users can browse them to view their creators (“-“ for the default session) and the number of users working within each. The user part presents user names, their groups, and their status (online, offline). The event part presents the list of events the user has received (users receive only those events with a level of importance equal to or greater than the level of awareness specified using the awareness menu).

Within the menus of the awareness tool the user can:

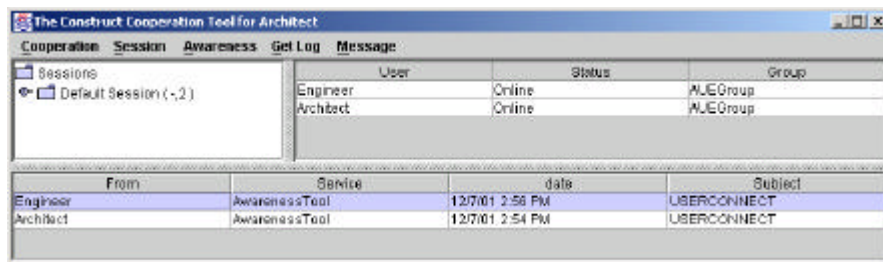
- Connect, disconnect, or exit the tool using the Cooperation menu;
- Create, join, leave, or close a session using the Session menu;
- Set the awareness level, or disable/enable awareness messages, using the Awareness menu. When the user wants to work in isolation, the user disables the awareness messages and when the user wants to work in a strong cooperation mode, the user enables them. The user sets the awareness level to a specific value when the user wants to receive messages with a fixed level of importance (e.g., normal and very important messages);
- Get the events produced before the user connected or only those related to a specific user, session, service, subject, or a date using the Get Log menu;
- Send/receive messages to/from other users using the Message menu.

The following screen shot shows the awareness tool of the engineer (Figure 6). The engineer realizes that the architect has already connected (2 minutes before him).

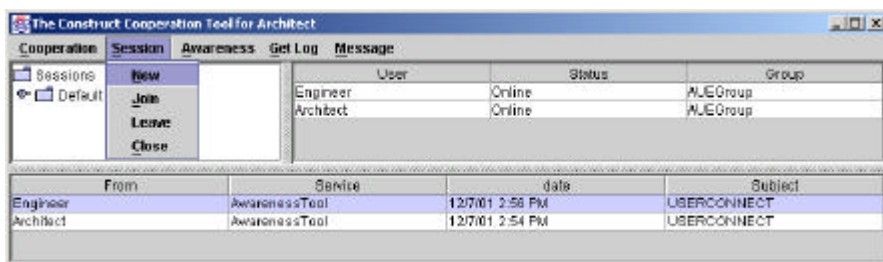


**Figure 6:** The awareness tool of the engineer after she has logged on.

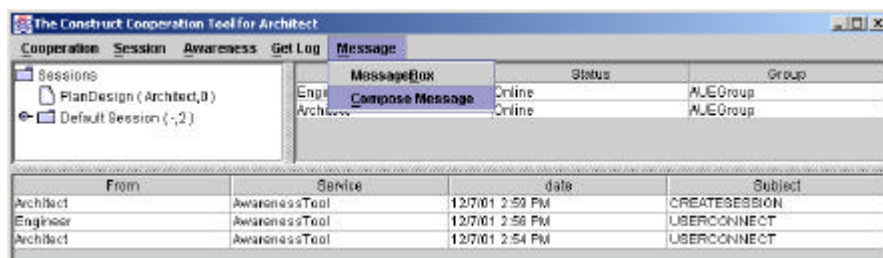
The architect is now aware that the engineer is connected (Figure 7): he received the event informing him that the engineer has been connected since 2:56 PM. He can now decide to cooperate with her on the design of the plan.

**Figure 7:** The awareness tool of the architect after the engineer has logged on.

The architect decides to create a new session that he will use to cooperate with the engineer (Figure 8).

**Figure 8:** The awareness tool of the architect as he is creating a new session.

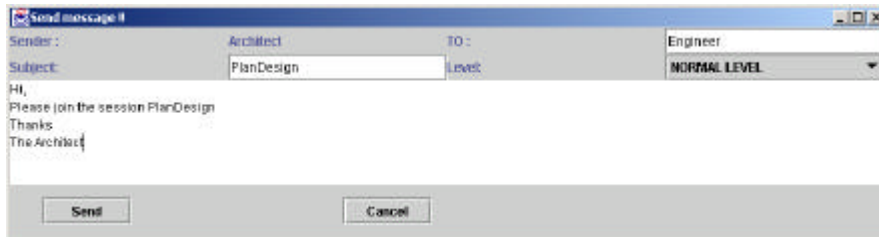
The architect has created a new session (see the event generated at 2:59 PM by the architect) and he will invite the engineer to join this session (Figure 9).





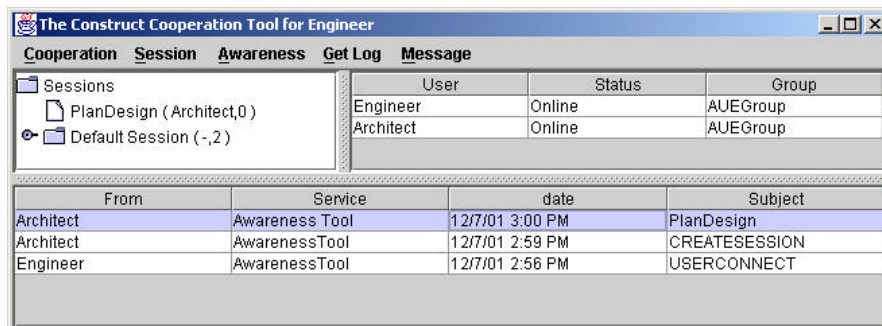
**Figure 9:** The awareness tool of the architect as he is creating a message to the engineer.

The Construct message tool (Figure 10) was developed to allow coworkers to send and receive messages inside the Construct environment.



**Figure 10:** The Construct message tool of the architect as he is sending a message to the engineer.

The engineer can now see (Figure 11) that the architect has created a session (see the event generated at 2:59 PM by the architect) and that the architect has sent her a message that she can read using her Construct message tool.



**Figure 11:** The awareness tool of the engineer after the architect has created a session and sent a message to the engineer.

The architect can now see (Figure 12) that the engineer has joined the session (event generated at 3:04 PM by the engineer) and that the economist has connected and has received all events sent by the architect and the engineer (events generated at 3:05 PM and 3:06 PM by the economist).

The Construct Cooperation Tool for Architect			
Cooperation Session Awareness Get Log Message			
Sessions			
PlanDesign (Architect,1)			
Default Session (-,3)			
User	Status	Group	
Engineer	Online	AUEGroup	
Architect	Online	AUEGroup	
Economist	Online	AUEGroup	
From	Service	date	Subject
Economist	AwarenessTool	12/7/01 3:06 PM	GETLOGALL
Economist	AwarenessTool	12/7/01 3:05 PM	USERCONNECT
Engineer	AwarenessTool	12/7/01 3:04 PM	JOINSESSION
Architect	AwarenessTool	12/7/01 3:59 PM	CREATESESSION
Engineer	AwarenessTool	12/7/01 2:56 PM	USERCONNECT
Architect	AwarenessTool	12/7/01 2:54 PM	USERCONNECT

**Figure 12:** The awareness tool of the architect after the engineer has joined the session and the economist has logged on and retrieved previous events.

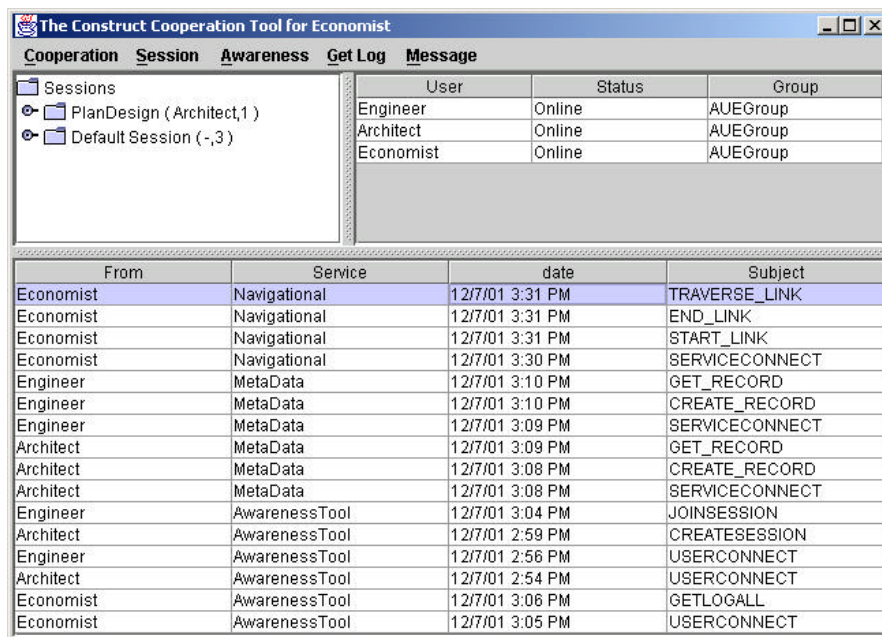
The engineer can now see (Figure 13) that the architect has generated metadata for the plan using the metadata structure service (events generated at 3:08 PM and 3:09 PM by the architect). She will use the metadata later to perform her work. This metadata may represent some rules (specifying some technique choices or the sketch of the plan) established by the architect. So she connects to the metadata service (event generated at 3:09 PM by the engineer) and gets the metadata related to the plan (event generated at 3:10 PM by the engineer).

The Construct Cooperation Tool for Engineer			
Cooperation Session Awareness Get Log Message			
Sessions			
PlanDesign (Architect,1)			
Default Session (-,3)			
User	Status	Group	
Engineer	Online	AUEGroup	
Architect	Online	AUEGroup	
Economist	Online	AUEGroup	
From	Service	date	Subject
Engineer	MetaData	12/7/01 3:10 PM	GET_RECORD
Engineer	MetaData	12/7/01 3:09 PM	SERVICECONNECT
Architect	MetaData	12/7/01 3:09 PM	GET_RECORD
Architect	MetaData	12/7/01 3:08 PM	CREATE_RECORD
Architect	MetaData	12/7/01 3:08 PM	SERVICECONNECT
Economist	AwarenessTool	12/7/01 3:06 PM	GETLOGALL
Economist	AwarenessTool	12/7/01 3:05 PM	USERCONNECT
Engineer	AwarenessTool	12/7/01 3:04 PM	JOINSESSION
Architect	Awareness Tool	12/7/01 3:00 PM	PlanDesign
Architect	AwarenessTool	12/7/01 2:59 PM	CREATESESSION
Engineer	AwarenessTool	12/7/01 2:56 PM	USERCONNECT

**Figure 13:** The awareness tool of the engineer after metadata operations of the architect and herself.

The economist might have waited for the generation of a first version of the plan to evaluate it. After some time she is informed by the architect that a

first version of the plan is generated. Just after the notification from the architect, the economist generates an estimate of the building construction cost corresponding to the current plan version. First, she creates the estimate using her favorite application and then she links the estimate to the plan. For this, she uses the navigational structure service. The events for these actions are generated at 3:30 PM and 3:31 PM (see Figure 14).



The Construct Cooperation Tool for Economist			
Cooperation Session Awareness Get Log Message			
Sessions		User	Status
PlanDesign (Architect,1)		Engineer	Online
Default Session (-,3)		Architect	Online
		Economist	Online
From	Service	date	Subject
Economist	Navigational	12/7/01 3:31 PM	TRAVERSE_LINK
Economist	Navigational	12/7/01 3:31 PM	END_LINK
Economist	Navigational	12/7/01 3:31 PM	START_LINK
Economist	Navigational	12/7/01 3:30 PM	SERVICECONNECT
Engineer	MetaData	12/7/01 3:10 PM	GET_RECORD
Engineer	MetaData	12/7/01 3:10 PM	CREATE_RECORD
Engineer	MetaData	12/7/01 3:09 PM	SERVICECONNECT
Architect	MetaData	12/7/01 3:09 PM	GET_RECORD
Architect	MetaData	12/7/01 3:08 PM	CREATE_RECORD
Architect	MetaData	12/7/01 3:08 PM	SERVICECONNECT
Engineer	AwarenessTool	12/7/01 3:04 PM	JOINSESSION
Architect	AwarenessTool	12/7/01 2:59 PM	CREATESESSION
Engineer	AwarenessTool	12/7/01 2:56 PM	USERCONNECT
Architect	AwarenessTool	12/7/01 2:54 PM	USERCONNECT
Economist	AwarenessTool	12/7/01 3:06 PM	GETLOGALL
Economist	AwarenessTool	12/7/01 3:05 PM	USERCONNECT

**Figure 14:** The awareness tool of the economist after she has created a link from the estimate to the plan.

This collaborative work process can continue until the completion of a final version of the plan with a cost estimate corresponding to the budget condition allocated to the construction of the building.

## 5. Open Issues and Future Work

This section discusses different open issues and future work plans relating to the overall task of developing cooperation services for Construct.

### 5.1 Placement of Cooperation Services within the Construct Architecture

Depending on the type of service they provide, new Construct services are integrated in the structure services layer or in the foundation services layer. The cooperation services discussed in this paper can be integrated into the Construct architecture in two different ways.

1. All messages in the environment can be routed through the cooperation services. This has the advantage that no existing services will have to be modified to operate as part of a collaborative environment. However, it also has some serious drawbacks regarding performance, as the cooperation services could become a bottleneck slowing the communication of messages. Also, this approach would create a single point of failure since all messages in the distributed environment must pass through one component.
2. Existing services can be updated to become aware of the cooperation services and make use of the provided facilities for cooperation. This solution maintains the distributed nature of the environment. However, one must deal with the issue of what services to update to make the environment use the cooperation services – should it be the wrappers or the structure services?

We quickly decided to go for the second option due to the distribution problems of the first option. Having decided on the second option, we had to address the associated issue. When integrating the cooperation services into Construct: should these new services be integrated into the foundation or the structure services layer or do we need a new layer for them?

**Foundation layer?** We could integrate cooperation services into the foundation services layer. This would require all structure services to be updated to make use of the cooperation services. Besides using the existing foundation services (e.g., the store), each structure service would need to communicate with the cooperation services to be part of a collaborative session. If, for instance, the navigational service were part of a collaborative session with two users, the navigational service would have to report its actions to the awareness service and potentially check for coordination rules

with some other cooperation service before performing its linking operations.

**Structure services layer?** A more natural place to integrate the cooperation services is the structure services layer considering the fact that cooperation services make use of the foundation services. For instance:

- ?? The session service uses the store service to save objects representing session information (session name, online users, etc.) and uses the access control service to control whenever a user can join a session;
- ?? The awareness service uses the notification control service to get and send events representing the activities that the cooperating users perform.

If the cooperation services are placed in the structure services layer, then the necessary updates to the architecture could be done either at the structure services layer or at the application layer (in the applications or in the wrapper services).

**New cooperation services layer?** We can also create a new layer (cooperation services layer – as depicted on Figure 3) considering the fact that cooperation services use the structure services and so have to be at a higher layer. In cooperative settings, users work together on shared objects and use several services within a session. In this context, the session service can use all structure services. With this solution, the necessary modifications to the architecture to use the cooperation services must happen at the application layer (in the applications or wrapper services).

Although the issue of what is the optimal placement of the cooperation services in Construct still remains open, we decided to aim for the last option in the first prototype and created a new layer for the session and awareness services. We modified the wrappers rather than the applications themselves to enable users of integrated applications to participate in cooperative work settings (like the one from the scenario in Section 4.2).

We plan to investigate this issue further in the future by creating more scenarios (and corresponding interaction diagrams) that use the cooperation

services. These interaction diagrams will then be the starting points for deeper analysis of the issue.

## 5.2 Adding Formal Cooperation Services

The session and awareness services can support a large number of cooperation types, especially those based on social protocols. Nevertheless, we need to extend this work in order to support more formal cooperation, allowing users to specify their roles and cooperation policies [31]. This is important especially in cooperative settings involving a large number of users and handling a large amount of data. It is also useful in this case to establish constraints for the cooperation to guide it to the common users' goal. To support this, we plan to extend Construct with a cooperation policy service based on access control and activity synchronization.

The access control, which can be provided by a Construct foundation service, supports users' roles. It defines for each cooperating user his/her capabilities. For example, granting *read* access to a user allows him/her to view the plan and gain the capabilities to play the role of an economist. Granting *write* access to a user allows him/her to draw the plan and get the capabilities to play the role of an engineer and/or an architect.

The activity synchronization, which can be provided by a new Construct foundation service, supports the coordination between cooperating users and the management of their responsibilities. In addition to the facilities provided by the access control, the activity synchronization defines constraints on cooperating users' interactions. For example, when two users *u1* and *u2* have *write* access to the plan, they can play the roles of the *engineer* and the *architect*. The coordination constraint "*u1 has to produce the final version of u2*" gives to the user *u1* the responsibility of the architect. In addition, this constraint coordinates the cooperation between *u1* and *u2* by defining a class of interaction scenarios in which *u1* and *u2* can update the plan as many times as they want, but *u1* is the last one who writes the plan.

The activity synchronization coordinates two cooperating users or activities according to a synchronization type. Synchronization types are inspired from research work done in several domains [4]. We note those done for advanced transactional models and those presented to support workflow interconnection paradigms [42]. These synchronization types can be classified into categories. In the first category, synchronizations are based

on shared data states (e.g., the first version of the data item  $d$  used by the activity  $B$  has to be the last version generated by the activity  $A$ ). In the second category, synchronizations are based on cooperating user activity states (e.g., the activity  $A$  has to be finished before the beginning of the activity  $B$ ).

### 5.3 Filtering for Structure Events in the Awareness Tool

We plan to experiment with additional filtering mechanisms in the awareness tool (cf. Section 4.1). We wish to investigate the possibilities of adding a filtering mechanism that can retrieve different categories of structural events (navigational, spatial, taxonomic, argumentation, etc.). Structural events may be of particular importance and relevance in a structural computing environment.

### 5.4 User Interface Design of the Cooperation Services

We plan to investigate issues relating to the user interfaces of the cooperation services – in particular the awareness tool. At present, two different types of user interface studies are planned. One study will evaluate the usability of the current user interface of the awareness tool. Another study will look into alternative ways to visualize the information presented in the awareness tool.

As part of this work, we plan to improve the user interface of the awareness tool in several ways. We have discovered a need to provide additional information about the users and groups. Both a role and a name and possibly also a picture should identify users. The role of groups should also be made more obvious in the user interface. We have also discovered the need to provide additional information about the operations in the Subject column such as what objects they are concerning and the location of these objects. Such information may be extremely useful in cooperative settings. Finally, the messages should be easier distinguishable from other events in the Subject column, e.g., by using different colors for different types of events and different levels of importance.

## 6. Related Work

Despite the fact that structural computing is a relatively new research area, important research results and prototypes have begun to emerge [23,27,28]. HOSS [24] was the first system to propose the new view of open hypermedia that formed the basis of the structural computing research area. HOSS advocated for the co-existence of several hypermedia domains within the same system. Since then a number of structural computing environments have emerged including Construct, Callimachus [18], FOHM [20], InfiniTe [1], and the system by Wang and Fernández [32].

One of the important contributions of Callimachus is the provision of development tools that support definition of new structural models based on the use of templates and structural patterns. FOHM defines a data model that supports interoperability at the data model level between three structural domains – taxonomic, navigational, and spatial. The system by Wang and Fernández presents an integrative design of graphical user interfaces of different structural domains. The design embraces features from navigational, spatial, taxonomic, and workflow domains. The work surrounding the InfiniTe system focuses on the application of structural computing technologies and techniques in the software engineering area.

To our knowledge, none of the related structural computing approaches has as of yet incorporated support for cooperation into their systems and environments. However, Construct and other systems addressing these issues can reuse results and experiences gained from incorporating cooperation support into open hypermedia systems (e.g., systems like SEPIA [14], DHM [11] and HyperDisco [39], as well as other related work [12,13]). We can also base our designs on well-known techniques from existing CSCW research systems [9,10] – in particular session management [7], awareness and coordination techniques [6,8,15,22], and transaction models [4].

One important lesson learned is that the trend towards more open and modular (component-based) environments makes design and implementation of cooperation services increasingly complex. This observation is based on experiences gained from more than a decade of developing collaborative hypermedia systems (i.e., HyperDisco [39,40] and EHTS [41]). Several aspects of modern component-based structural computing environments add to this increased complexity:



- ?? The aspect that data and structure is stored separately. Construct is only in control of the structure. Data is stores elsewhere, potentially in data stores that provide no support for cooperation. This makes it difficult to provide coherent cooperation support across both the data and structure domain.
- ?? The aspect that most of the applications in Construct are third party applications that have been integrated with the environment. These third party applications may not support cooperation and may not be easy to extend to support cooperation. Sometimes the wrapper service (if one exists) can help compensate for this.
- ?? The aspect that each service is wrapped in its own component. The result is that the cooperation support is spread over several service components. This adds cognitive overhead in the design of the cooperation services and adds communication overhead at runtime.
- ?? The aspect that each service component operates in a layered architectural framework. As discussed in section 5.1 it is difficult to position the cooperation services within the service layers of the architecture. Several options exist and each option has its advantages and drawbacks.

## 7. Conclusion

This paper presents the design and implementation of cooperation services in Construct. At present, we have implemented a session service, an awareness service, and an awareness tool allowing each user to see, according to their selected awareness level, the services that other users are using and events related to operations they are performing. The user can also set their awareness level and get events about operations that other users have performed before their login. The user can in addition send/receive messages to/from other users. These tools and services allow cooperating users to maintain awareness of each other.

The cooperation services have been integrated into Construct. The integration of the cooperation services allows new types of cooperative work practices to be supported using Construct. While the integration of cooperation services has added to the overall capability of Construct, it has also uncovered several issues that must be dealt with in future work.

We think that this work constitutes a first important step toward the development of a full suite of cooperation services and consequently toward supporting a wide variety of cooperative work practices with Construct.

### Acknowledgments

We wish to thank Monica Schraefel (user interface design) and Sigi Reich (filtering for structural events) for valuable comments at the Third International Workshop on Structural Computing (SC3) that helped shape the presentation of this paper. The present paper is an expanded version of a paper presented at SC3 [30]. We also wish to thank the anonymous reviewers for providing helpful comments.

### References

1. K. M. Anderson and S. A. Sherba. Using structural computing to support information integration. In *Proceedings of the Third International Workshop on Structural Computing*, Århus, Denmark, August 2001.
2. T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The World Wide Web. *Communications of the ACM*, 37(8):76-82, August 1994.
3. V. Bush. As we may think. *Atlantic Monthly*, 176(1):101-108, July 1945.
4. P. K. Chrysanthis and K. Ramamritham. ACTA: The SAGA continues. A. Elmagarmid (Eds.) *Transaction models for advanced database applications*. Morgan-Kaufmann, 1992.
5. Construct. 2002. <http://www.cs.aue.auc.dk/construct>
6. P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 51-58, Toronto, Canada, November 1992.
7. W. K. Edwards. Session management in collaborative applications. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 332-330, Chapel Hill, NC, USA, October 1994.
8. W. K. Edwards. Coordination infrastructure in collaborative systems. Ph.D. thesis, Georgia Institute of Technology, College of Computing, Atlanta, GA, USA, December 1995.
9. C. A. Ellis, A. Gibbs, and G. Rein. Groupware: Some issues and experiences. *Communications of the ACM*, 34(1): 38-58, January 1991.

10. R. S. Fish, R. E. Kraut, and M. P. Leland. Quilt: A collaborative tool for cooperative writing. In *Proceedings of the Conference on Office Automation Systems, Collaborative Work*, pages 30-37, 1988.
11. K. Grønbaek, J. A. Hem, O. L. Madsen, and L. Sloth. Cooperative hypermedia systems: A Dexter-based architecture. *Communications of the ACM*, 37(2):64-74, February 1994.
12. J. M. Haake, U. K. Wiil, and P. J. Nürnberg. Openness in shared hypermedia workspaces: The case for collaborative open hypermedia systems. *SIGWEB Newsletter*, 8(3):33-45, October 1999.
13. J. M. Haake. Structural computing in the collaborative work domain. In *Proceedings of the Second International Workshop on Structural Computing*, Lecture Notes in Computer Science (LNCS 1903), pages 147-159, Springer Verlag.
14. J. M. Haake and B. Wilson. Supporting collaborative writing of hyperdocuments in SEPIA. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 138-146, Toronto, Canada, October 1992.
15. J. M. Haake. Facilitating orientation in shared hypermedia workspaces. In *Proceedings of the ACM Conference on Supporting Group Work*, pages 365-374, Phoenix, AZ, USA, November 1999.
16. D. L. Hicks and U. K. Wiil. Towards collaborative design in an open hypermedia environment. In *Proceedings of the Sixth International Conference on CSCW in Design*, London, Canada, July 2001.
17. R. D. Hill, T. Brink, F. Patterson, S. L. Rohall, and W. T. Winter. The rendezvous language and architecture. *Communications of the ACM*, 36(1):62-67, January 1993.
18. M. Kyriakopoulou, D. Avramidis, M. Vaitis, M. Tzagarakis, and D. Christodoulakis. Broadening structural computing systems towards hypermedia development. In *Proceedings of the Third International Workshop on Structural Computing*, Århus, Denmark, August 2001.
19. C. C. Marshall and F. M. Shipman. Spatial hypertext: Designing for change. *Communications of the ACM*, 38(8):88-97, August 1995.
20. D. E. Millard and H. C. Davis. Navigating spaces: The semantics of cross domain interoperability. In *Proceedings of the Second International Workshop on Structural Computing*, Lecture Notes in Computer Science (LNCS 1903), Springer Verlag.
21. Y. Neveu, Y. Guervilly, U. K. Wiil, and D. L. Hicks. Providing metadata services on the World Wide Web. Technical Report CSE-01-01, Department of Computer Science and Engineering, Aalborg University Esbjerg, Denmark, March 2001.

22. T. Nomura, K. Hayashi, T. Hazama, and S. Gudmundson. Interlocus: Workspace configuration mechanisms for activity awareness. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 19-28, Seattle, WA, USA, November 1998.
23. P. J. Nürnberg (ed.). *Proceedings of the First Workshop on Structural Computing*. Technical Report AUE-CS-99-04, Aalborg University Esbjerg, Denmark, 1999.
24. P. J. Nürnberg, J. J. Leggett, and E. R. Schneider. As we should have thought. In *Proceedings of 1997 ACM Conference on Hypertext*, pages 96-101, Southampton, UK, April 1997.
25. P. J. Nürnberg, J. J. Leggett, E. Schneider, and J. Schnase. HOSS: A new paradigm for computing. In *Proceedings of the 1996 ACM Conference on Hypertext*, Washington, DC, USA, March 1996.
26. H. Van Dyke Parunak. Don't link me in: Set based hypermedia for taxonomic reasoning. In *Proceedings of the 1991 ACM Conference on Hypertext*, pages 233-242, San Antonio, TX, USA, December 1991.
27. S. Reich, M. M. Tzagarakis, and P. De Bra (eds.). *Proceedings of the Third International Workshop on Structural Computing*, Lecture Notes in Computer Science, Springer Verlag, 2001.
28. S. Reich and K. M. Anderson (eds.). *Proceedings of the Second International Workshop on Structural Computing*, Lecture Notes in Computer Science (LNCS 1903), Springer Verlag, 2000.
29. P. Smolensky, B. Bell, B. Fox, R. King, and C. Lewis. Constraint-based hypertext for argumentation. In *Proceedings of the 1987 ACM Conference on Hypertext*, pages 215-245, Chapel Hill, NC, USA, November 1987.
30. S. Tata, D. L. Hicks, U. K. Wiil. Cooperation services in a structural computing environment. In *Proceedings of the Third International Workshop on Structural Computing*, Århus, Denmark, August 2001.
31. S. Tata, G. Canals, and C. Godart. Specifying interactions in cooperative applications. In *Proceedings of the 11<sup>th</sup> International Conference on Software Engineering and Knowledge Engineering*, Kaiserslautern, Germany, June 1999.
32. W. Wang and A. Fernández. A graphical user interface integrating features from different hypertext domains. In *Proceedings of the Third International Workshop on Structural Computing*, Århus, Denmark, August 2001.
33. U. K. Wiil and D. L. Hicks. Providing structural computing services on the World Wide Web. In *Proceedings of the Third International Workshop on Structural Computing*, Århus, Denmark, August 2001.

34. U. K. Wiil. Development tools in component-based structural computing environments. In *Proceedings of the Seventh Workshop on Open Hypermedia Systems*, Århus, Denmark, August 2001.
35. U. K. Wiil, D. L. Hicks, and P. J. Nürnberg. Multiple open services: A new approach to service provision in open hypermedia systems. In *Proceedings of the 2001 ACM Conference on Hypertext*, Århus, Denmark, August 2001.
36. U. K. Wiil. Using the Construct development environment to generate a file-based hypermedia storage service. In *Proceedings of the Second International Workshop on Structural Computing*, Lecture Notes in Computer Science (LNCS 1903), pages 147-159, Springer Verlag.
37. U. K. Wiil and P. J. Nürnberg. Evolving hypermedia middleware services: Lessons and observations. In *Proceedings of the 1999 ACM Symposium on Applied Computing*, pages 427-436, San Antonio, TX, USA, February 1999.
38. U. K. Wiil. Open hypermedia: Systems, interoperability and standards. *Special Issue on Open Hypermedia, Journal of Digital Information*, 1(2), December 1997.
39. U. K. Wiil and J. J. Leggett. The HyperDisco approach to Internet distribution. In *Proceedings of the 1997 ACM Conference on Hypertext*, pages 13-23, Southampton, UK, April 1997.
40. U. K. Wiil and J. J. Leggett. The HyperDisco approach to open hypermedia systems. In *Proceedings of the 1996 ACM Conference on Hypertext*, pages 140-128, Washington, DC, USA, March 1996.
41. U. K. Wiil. Issues in the design of EHTS: A multiuser hypertext system for collaboration. In *Proceedings of the 25<sup>th</sup> IEEE Hawaii International Conference on System Sciences*, pages 629-639, Kauai, HI, USA, January 1992.
42. Workflow Management Coalition. 2002. <http://wfmc.org>