# Livespaces

## Technology Overview

Matthew Phillips

Defence Science and Technology Organisation

Australian Government

Department of Defence

Defence Science and
Technology Organisation

# Abstract

This paper describes Livespaces, a technology framework developed by DSTO to support advanced meeting spaces and distributed multi-site collaboration. It discusses the rationale behind the Livespace concept, the history of the research and development that led to the Livespaces approach, and, in particular, its roots in providing support for the intense collaboration sessions often required by ADF operational planning specialists. The novel technical architecture employed by the Livespaces operating environment is described, as well as the new capabilities it enables. The paper also discusses possible configurations for a Livespace and various applicable off-the-shelf hardware technologies and their trade-offs.

# Scope

This is a technical paper, intended for readers that wish to understand what a Livespace is, what it provides and how the environment that supports it operates. It provides a detailed discussion of the techniques that the environment uses to implement a distributed collaboration framework, as well as the component services that collectively comprise a Livespace. It also discusses some of the experimental group-ware applications built using the software framework and how various commercial hardware components can be employed to build a Livespace.

# Executive Summary

This report contains a technical definition of the Livespaces technology framework developed by DSTO to support advanced meeting spaces and distributed multi-site collaboration. It defines the rationale behind the development of the Livespace concept, and describes the capabilities a Livespace provides.

The Livespace concept originates from DSTO research into supporting the intense collaboration sessions often required of ADF planning specialists during the initial planning phase of an operation. Initial experiments in supporting this type of collaboration were carried out using a prototype assembled from components developed by several 3rd parties, including Stanford and the Distributed Systems Technology Centre. These led to an initial Livespace prototype room at The University of South Australia, which was trialled as part of a series of Technical Exercises involving ADF planning staff. The requirements and experiences arising from the prototype were incorporated into the novel architecture for developing collaborative meeting spaces which is described in this report.

This report describes the requirements and technical problem space that led to the design of the Livespace Bus, a distributed systems approach to solving the problem of integrating a disparate and distributed set of software and hardware components into a single manageable system. It includes a description of how the bus operates and provides a brief example showing how a software developer can use this framework to rapidly develop new Livespace services or extend existing ones.

The software applications and services that have been developed on top of the Livespace Bus framework are described: these include various experimental group-ware applications and the desktop applications for managing a Livespace smart meeting room, such as environmental settings (lights, volume, video switching, etc.)

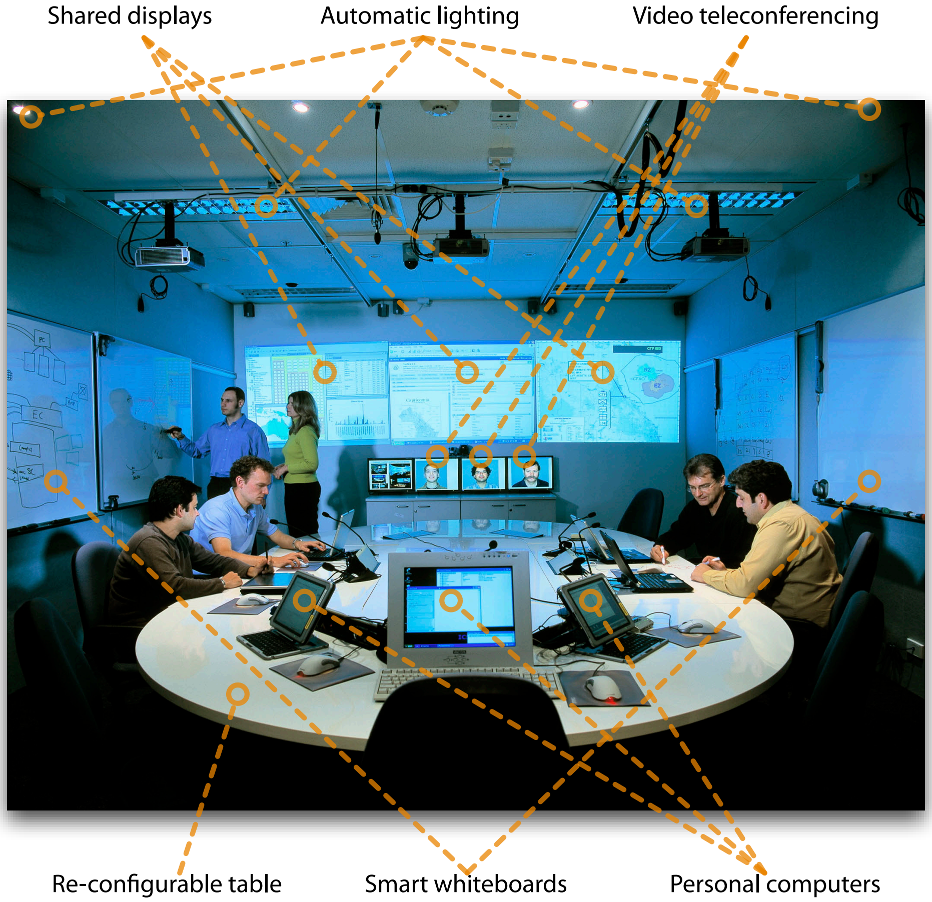
The report also discusses the various approaches to setting up a Livespace, its layout, trade-offs, and the various applicable hardware technologies that may be employed.

In the future work section, we highlight the fact that Livespaces has matured to a level of stability that has enabled it to be successfully deployed to a number of ADF sites for advanced trials. It has also been deployed by Canada's DRDC to three Canadian sites under a TTCP Materiel Transfer Agreement. We discuss options for expanding the scope of Livespace application and collaboration, and recommend that an effective approach would be to make the Livespace framework available under an open source license.

# 1. What Is A Livespace?

A Livespace is a technology-enhanced collaboration space for a team of people. The purpose of a Livespace is to integrate technologies that help people work together: to bring these technologies together into a supporting system that becomes part of the background, rather than the more common situation where these technologies appear as a set of disparate, idiosyncratic and quirky hardware gadgets and software applications.

The Livespace approach is most effective in advanced meeting spaces equipped with a range of capabilities, since it's in these sorts of spaces that the cognitive overload of managing the devices and applications becomes most significant. However, the application of the Livespace concept is not tied to a particular set of technologies: it can be applied equally to a meeting room full of the latest experimental and off-the-shelf computing systems and audio-visual equipment, or to a collection of personal laptops temporarily networked together for an ad-hoc meeting.



*DSTO's Intense Collaboration Space*

## 1.1. The Intense Collaboration Space

A prime example of what we mean by a Livespace is DSTO's Intense Collaboration Space (ICS). This is a meeting space designed to enable prototyping and experimentation to support the sort of intense collaboration that occurs between small groups of military specialists during the planning phase of an military operation.

The ICS is oriented around a re-configurable central conference table, with small form-factor PCs for each participant, three shared projected displays in the front, smart whiteboards, and supported by a number of background capabilities such as video teleconferencing, video and audio switching, and automatic lighting control.

One way to get a feel what a Livespace provides is to describe how it is used from a meeting participant's point of view. The next section runs through a general scenario of use.

## 1.2. A User's View Of A Livespace

The participants in a Livespace session will find a control panel inside the door that enables them to activate the room: this will turn on the room's various devices (lights, projectors, etc.) and initialise any background software applications. Each user will usually log on to a personal computer on the central table. As well as standard desktop software, the personal computers run a range of custom collaborative software which provides facilities for shared document editing, document management, text messaging, and applications for controlling the room itself, such as lighting and volume.

As well as personal displays, the users also have access to three large shared computing spaces projected onto the front wall. These displays, which can be controlled from any of the participant's personal PCs, provide a shared display space for information, such as documents, web pages or video. There may also be a number of smaller 'ambient' shared displays (typically LCD screens) in the room that can be used for displaying background information, such as the meeting agenda. At any time the display from any of the personal computers can be shared by switching it up to the large display area. The participants can also sketch on one of the room's whiteboards – a capture device attached to the whiteboard digitises these sketches and shows them in an application on one of the shared displays.

The room also has telepresence capabilities for collaboration with groups in remote Livespaces. These capabilities include conventional video teleconferencing, document sharing and remote screen sharing, which enables participants to see and interact with remote displays. The room's collaborative applications can be connected to support the telepresence collaboration.

At the end of a session, any artefacts generated, such as documents, minutes and sketches, can be stored to a document repository. The same control panel that activated the room is then used to turn it off and leave it in a state ready for its next session.

# 2. Why Do We Need Livespaces?

The driving force for Livespace research is to reap what believe is a largely untapped potential for technology to significantly advance group collaboration. The current explosion of Internet-based communication technologies has clearly demonstrated how these innovations could significantly change the way groups collaborate – yet if you walk into a typical corporate meeting room today, you're unlikely to find any collaboration technology more sophisticated than a video teleconferencing unit. While there is no lack of innovative technology to choose from, we're making less progress than we'd like in effectively employing it.

There are many low-cost collaboration technologies becoming available, including an embarrassing number of instant messaging networks, VoIP-based internet telephony and video conferencing. These new consumer products are feeding a growing public awareness of the possibilities. However many of the popular collaboration technologies are solely consumer-focused, often restricted to one-to-one communication between two PCs and mediated by proprietary servers.

In addition, due to their relative immaturity, many collaboration technologies are built inside proprietary walled gardens, making them difficult to integrate, as well as inflexible and idiosyncratic. Unlike established technologies, such as email, these technologies tend to have their own interfaces, protocols and authentication, which all cumulatively increase the friction involved in using them. It is a sad fact that any given group of people can be confident of being able to use 'old' technologies such telephone or email in their daily work, but relying on anything more advanced usually incurs an unacceptable risk of technical problems dominating over productive work.

The Livespace goal is to meld available technology into a space in which people feel comfortable and confident. A successful Livespace should make taking advantage of technology effortless, rather than being an ongoing battle against a conglomeration of recalcitrant gadgets. At a minimum, achieving this goal will involve moving our conceptualisation a step up from a PC and application-focused approach toward treating the meeting space itself as the application.

A large part of a Livespace's function is to make hardware invisible. If a user has to hunt for a remote control or open a computer rack, then the Livespace has failed. Perhaps the number of remote controls for the devices in the ICS alone is the most telling indication of how futile it would be to simply drop these technologies into a room with no further thought.

*A daunting set of controls from a meeting room*

While it's unlikely readers will disagree with these sentiments – indeed they verge on being motherhood statements – the reason that 'recalcitrant gadgets' are often a sad fact of meeting spaces today is not due to a lack of trying to achieve the above goals. It seems that even people who are expert in this area have inherent difficulty defining conceptual models that support advanced collaboration without exposing the complexity of the technology underlying it. While we certainly don't claim to have solved this problem, the Livespaces project is working toward a new approach creating effective collaboration spaces: the next sections describe how we've gone about it.

# 3. How To Create A Livespace

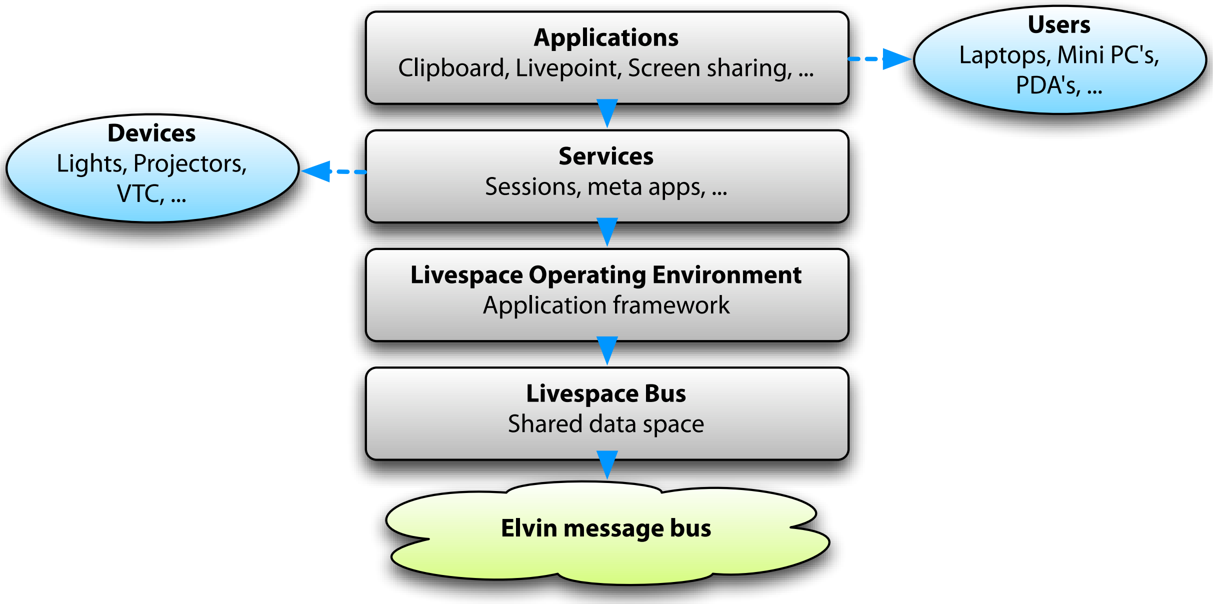
Many areas of research and development need to be addressed to achieve our goals for a Livespace. This paper focuses on the architectural aspects of a Livespace: the infrastructure needed to integrate the core components that we envisage in a Livespace. Most of the required original research falls in the areas of human-computer interaction (HCI) and situational awareness: these aspects of a Livespace are being researched in parallel to this work by DSTO and its partners in HxI, using the technology infrastructure described here as a testbed.

A Livespace needs to integrate a wide range of hardware and software: lights, projected displays, audio, video, information storage, and a milieu of software applications. In fact, in many ways, a Livespace needs to act as an operating system for the meeting space and, like an operating system, the natural way to approach managing such a disparate array of components is start by defining a flexible plug-and-play framework into which the components can be integrated.

The core component of a Livespace that provides this distributed plug-and-play framework is termed the *Livespace Bus*: in the operating system metaphor, this plays the role of the peripheral device bus. All services and applications in a Livespace room use this bus to discover and communicate with each other.

## 3.1. The Livespace Architecture

The diagram below places the Livespace Bus in context, showing how it supports the other core components of a Livespace.



*The Livespace architecture stack*

Working from the bottom up, we have the following core components:

**Elvin message bus**. At the lowest level of the Livespace stack is the Elvin publish/subscribe message bus (Segall *et al.* 2000). Elvin provides an abstraction over point-to-point network communications, enabling clients to broadcast and receive messages without necessarily being aware of each other. Elvin clients need simply subscribe to messages of interest using content-based selection criteria, and Elvin takes care of routing those messages to them when they're published somewhere on the bus. Elvin routers can also be federated into efficient wide-area messaging networks, which is key to supporting the connection of Livespaces across sites.

**Livespace Bus**. The Livespace Bus is another level of communications abstraction: it defines a shared data space of discoverable, modifiable data objects called *entities*. The Livespace Bus entity model is fundamental to a Livespace, and is used for many purposes, including device discovery and coordination, and application-level information management.

**Livespace Operating Environment**. This is a framework for building Livespace services and applications. It includes a component model for packaging and deploying software components, and a centralised service management and configuration system.

**Services**. This represents the set of core services that drive a Livespace. This includes services for controlling a room's hardware devices (lights, projectors, audio/video switching, video conferencing), switching between pre-set room configurations (sessions), multimedia display, and high-level information systems such as the room-wide shared clipboard and information repository. In the operating system metaphor we've been using, the services that manage a room's hardware resources can be thought of as the equivalent of device drivers.

**Applications**. These are desktop applications such as the Livepoint remote mouse control system, remote screen sharing, instant messaging, group brainstorming and the Ignite room control system.

The core applications provided by a Livespace include:

**Ignite**. The main user interface to the room's devices and services: this includes lighting, audio and video switching and applying pre-defined commonly-used room configurations.

**Screen sharing**. Provides the ability to push a live copy of a computer's screen onto another computer.

**Link sharing**. Allows users to quickly open links to web pages and documents on remote displays.

**Livepoint**. Provides the ability to control remote screens as if they were attached to the local computer.

**TeamThink**. A shared document editor that can be used to collaboratively edit a live document, often used during brainstorming sessions.

**Sticker.** Provides instant messaging between individuals or to the entire room, as well as displaying news feeds.

These Livespace applications, as well as others that support room management, are described in more detail later in this paper.

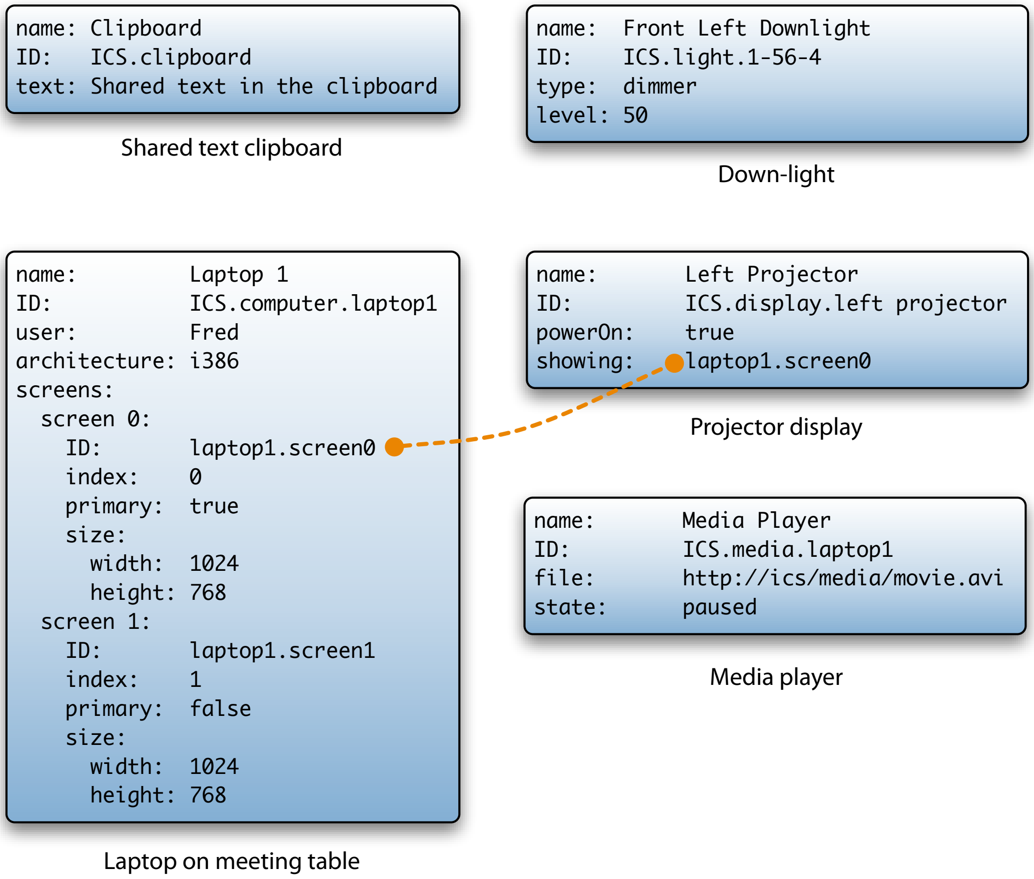
Before we show how the Livespace services and applications work together, we will need to describe the Livespace Bus in more detail, since the Livespace components all operate in terms of the abstractions it provides.

## 3.2. The Livespace Bus

The Livespace Bus is a space of distributed data objects, or *entities*. An entity is simply a data resource that is published into the shared space where it can be discovered, monitored and modified. Services, whether they be low-level hardware drivers, or high-level information resources, publish an entity-based model of themselves on the bus where clients can discover them. As the underlying device or information resource changes, the entities are kept synchronised. Equally, when a client changes an entity, the underlying service tries to 'make it so' by making the equivalent change to the real resource the entity represents.

One way to illustrate this approach is via an example of some of the entity-based services in a Livespace. In the diagram below examples of four different entity types are shown: we'll run through each one from simplest to most complex.[1]

[1] For the purposes of the example some of the example entities have been simplified from those deployed in an operational Livespace.



```
name: Clipboard
ID:   ICS.clipboard
text: Shared text in the clipboard
```

Shared text clipboard

```
name:  Front Left Downlight
ID:    ICS.light.1-56-4
type:  dimmer
level: 50
```

Down-light

```
name:          Laptop 1
ID:            ICS.computer.laptop1
user:          Fred
architecture:  i386
screens:
  screen 0:
    ID:        laptop1.screen0
    index:     0
    primary:   true
    size:
      width:   1024
      height:  768
  screen 1:
    ID:        laptop1.screen1
    index:     1
    primary:   false
    size:
      width:   1024
      height:  768
```

Laptop on meeting table

```
name:      Left Projector
ID:        ICS.display.left projector
powerOn:   true
showing:   laptop1.screen0
```

Projector display

```
name:        Media Player
ID:          ICS.media.laptop1
file:        http://ics/media/movie.avi
state:       paused
```

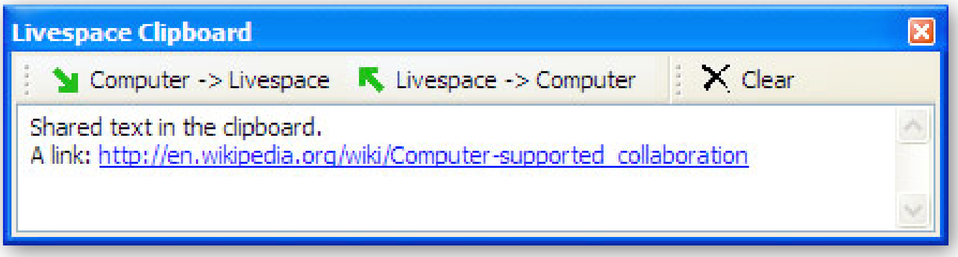
Media player

*Some example Livespace entities*

## Example: Clipboard

Every entity has at least two fields associated with it: a globally-unique ID and a human-readable name. An entity will also have a number of type-specific fields, which may be values such as text, numbers or dates, or nested sub-entities.

The simplest entity in a Livespace is the Clipboard. This has just one field, `text`, that holds text that is available to the whole Livespace. This entity is presented to the user via a simple desktop applet that allows the user to quickly edit the shared text, and copy it in and out of the local clipboard.



Shared text in the clipboard.
A link: http://en.wikipedia.org/wiki/Computer-supported_collaboration

*The Livespace clipboard window*

### Example: Light

Unlike the clipboard, which is a passive container for text, the Light entity represents the state of an active device in the room: a light source. The Light entity's most important field is its `level`, which represents the light's current illumination level: changing the level causes the service that published the entity to issue the commands needed to make the light follow suit.

### Example: Media Player

The media player entity is an example of a common modelling pattern which employs a metaphor from a real device. The media player has two key properties, `file` and `state`: to play a media file we first put a link to the file in the `file` property, and then set `state` to 'playing'. Once playing, we can set the state to 'paused' or 'stopped' to control playback. This entity model mimics a real device, such as a CD player, where the user loads a disc and then sets the state using the play and pause buttons on the device.

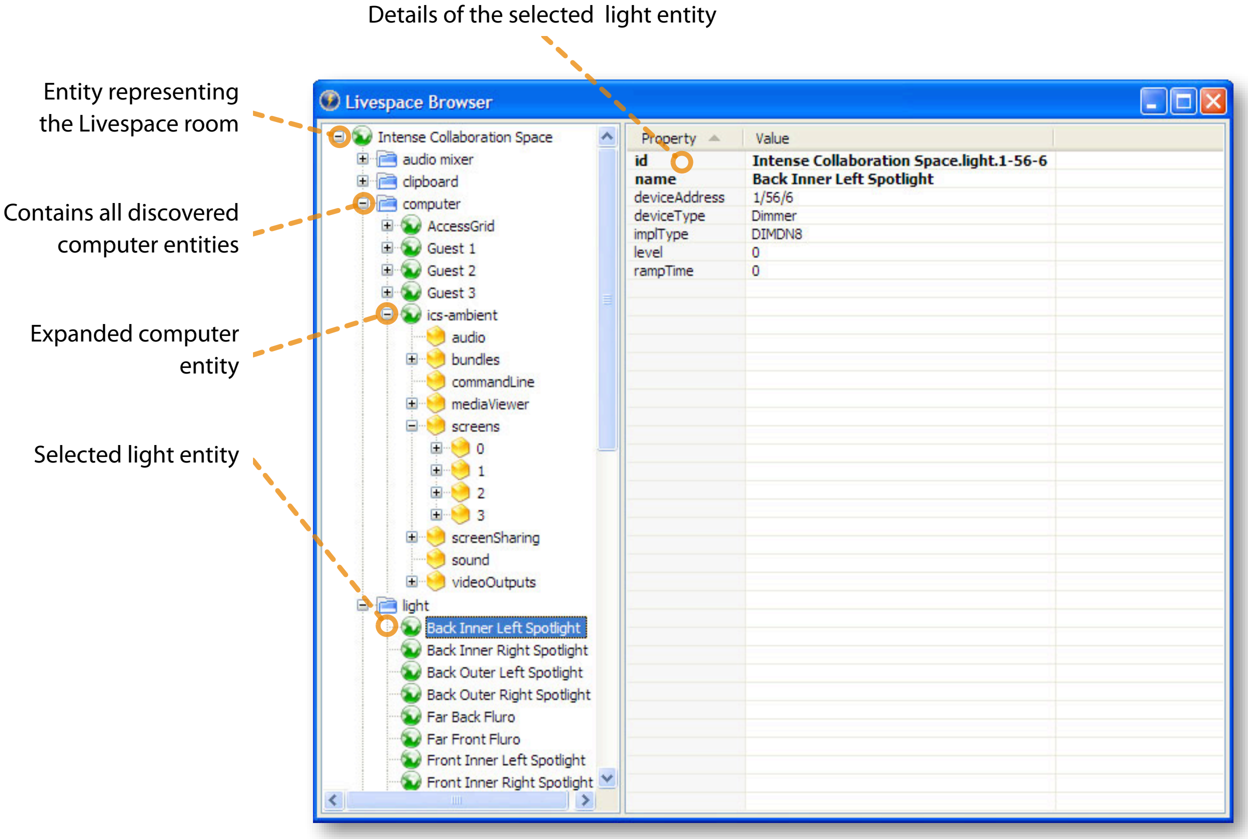
### Example: Computer

The Computer entity is one of the more complex entities in the space, and is used by many other services in the room. One of these entities is published by each computer in a Livespace – in the example above, we show a subset of the information, which includes the computer's name, the currently logged in user and its CPU architecture. The diagram also shows the information a computer publishes about its video displays: the example computer has two desktops on two screens running at 1024 x 768 pixel resolution.

The example diagram also shows a logical link between one of the computer's screens and a Display entity that represents a video projector in the room. This link is established simply by putting the screen's ID into the Display's `showing` field: when this field changes, the service in the room that controls the room's video switch synchronises by switching the video source from the computer's first screen to the projector's video input. The device driver analogy is apt in this example: Livespace clients only need to know how to change the correct entities control video switching, only the back-end service needs to know how to actually control the video switch hardware and look up the associated input and output video ports.

## 3.3. Visibility And Understanding Via Browsing

Because the Livespace Bus is built on a space of uniform distributed resources, the space itself can be browsed and manipulated without any *a priori* knowledge of its structure. One key benefit this gives is the ability to browse and manipulate the entities in the space, a facility which is provided by the Livespace Browser application. The screen capture below shows an example of the Livespace Browser in the Intense Collaboration Space.



Details of the selected light entity

Entity representing the Livespace room

Contains all discovered computer entities

Expanded computer entity

Selected light entity

| Property | Value |
|---|---|
| **id** | **Intense Collaboration Space.light.1-56-6** |
| **name** | **Back Inner Left Spotlight** |
| deviceAddress | 1/56/6 |
| deviceType | Dimmer |
| implType | DIMDN8 |
| level | 0 |
| rampTime | 0 |

*The Livespace Browser application*

The Livespace browser follows the common metaphor of showing an expandable hierarchy on the left and the detail of the selected item on the right. The root of the hierarchy is the entity representing the room itself: in this case it's DSTO's Intense Collaboration Space. Under this are a series of folders for each entity type: as these folders are expanded, the entities of that type appear as children.

In the example, the `computer` and `light` entity type folders have been expanded, and the user has further expanded the detail of the `ics-ambient` computer, showing some of its internal structure. The user has selected the `Back Inner Left Spotlight` light entity, and its properties appear in the detail pane on the right. You can see that the light's level is currently at '0', meaning the light is off: the user could, if they wish, enter a new value such as '100' directly into the browser to turn the light on.

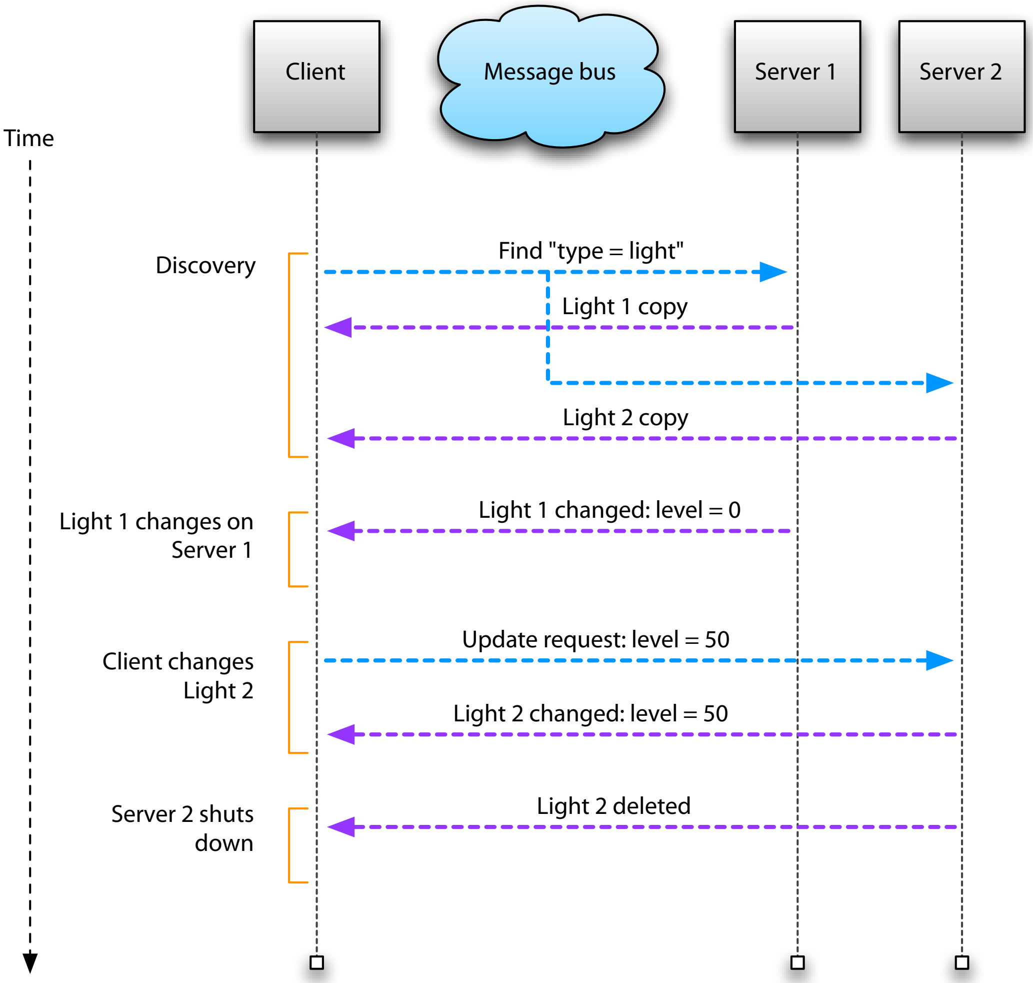
The ability to browse and manipulate the state of a Livespace in this way has proven extremely useful for developers and administrators and we believe it is one of the key advantages to this kind of approach over the more opaque RPC-style systems discussed later.

## 3.4. Replication And Synchronisation

The Livespace Bus is implemented using a dynamic entity replication system. On the bus there are two types of agent: servers, which publish entities, and clients, which discover and maintain local replicas of them. Although the client/server terminology may suggest a significant difference, a server is actually very similar to a client: they both host a pool of local entities that can be accessed and changed, a server simply has the additional role as the publisher of the master copy of its entities.

Once a client has discovered an entity, it maintains a local copy of it for the duration of its interest, and will update it when the server notifies it of changes to the master copy. Any changes made on the client's replica are visible immediately on the client, while at the same time a change request is sent asynchronously to the server. This is an optimistic, lock free system in which a client initially assumes that an update will succeed – if a change is later vetoed by the server, the client must deal with rolling back the change. If a change is not acknowledged, the client must assume that the server is down, and will act as if the entity had been deleted. At any given time, client and server may be out of sync, however the system does guarantee that they will eventually reach a synchronised state.

All the communications between clients and servers is via the Elvin event notification system, which takes responsibility for routing messages efficiently between any number of event producers and consumers. To illustrate how the Livespace Bus uses the message router for entity replication, we'll run through the following example. In the sequence diagram below, we list the sequence of events that occur when a client discovers two `light` entities, monitors a change from the server, and then makes a change itself.



*Entity replication sequence*

The Livespace messages transferred over Elvin come in three categories: *find*, *update* and *info*. The *find* message type is used to query for entities hosted by server containers. The *update* message is emitted by a client to request a state change to an entity. The *info* message type contains entity state, and is emitted by server containers in response to *find* requests, or when the entity changes.

An *info* message either contains the entire state of an entity, or the delta between the previous state and the new one.[2] Thus the initial Discovery phase shown in the diagram above would involve the client sending a single *find*, followed by receipt of two *info* replies from the servers, each containing the full state of a light entity. The following change of Light 1's level would contain only a delta to the `level` value from the previous *info*.

One of the chief advantages of this architecture is that reading and writing to entities is a non-blocking operation: changes are always made instantly on the local copy, while the request/response transaction to make it actually happen is handled asynchronously. The replication model also means there is no necessary distinction between client and server entities: Livespace programs access an entity in the same way regardless of whether it is the master copy or a client replica. In fact, entities do not have to be managed by a client or server at all, making testing easier by allowing developers to test against off-line 'dummy' entities.

One of the chief disadvantages of this approach is the fact that clients must be prepared to deal with rollbacks to changes that initially appeared to be successful: this may happen due to a service vetoing the request, the change being trumped by another client, or if the server has silently failed and the request is not acknowledged. In practice this has not turned out to be much of an issue, since clients are typically programmed in an event-driven fashion anyway, and these abnormal situations are indicated using the same notification route as 'normal' events: if a server vetoes a change (e.g. a request to change a light level to '-1'), the client simply sees the light's level change to '-1' and then change back to the initial value, at which point it deals with it in the same way as any other kind of change. In the case where a server has abruptly failed, the client will be notified that the entity was deleted when a liveness check fails, and deal with it as with a normal delete.

One other disadvantage of this approach is the fact that clients and servers can be out of synchronisation for a small window of time. This has the implication that clients on different machines with replicas of the same entity may also temporarily disagree about its state. While there may be good reasons to argue that this distributed systems version of general relativity is a fundamental property of such systems, it is also true that for applications with harder requirements for synchronisation, this would not be sufficient. However, the applications in a Livespace do not have such stringent requirements, and we have found in practice that this approach represents a good compromise.

## 3.5. The Livespace Bus Development Model

From the developer's perspective, entities are simply pure-data objects that can be read, written to, and monitored for changes. By 'pure-data' we mean that an entity has no behaviour or methods that can be

[2] The entity state in *info* messages is encoded directly into the Elvin notification format where possible, but since Elvin notifications do not support compound data structures (values in a notification must be numbers, strings or binary byte arrays), an XML transfer format is used as a fallback for more complex structures.

invoked aside from those that read and write its state. In fact, there are only three logical operations that can be invoked on an entity:

**get value**: read the value of a property. For example: `light.getValue (“level”)`

**set value**: set the value of a property. For example: `light.setValue (“level”, 50)`

**listen**: register to be notified whenever the state of the entity changes. A listener receives the name of the property and its old and new values. Since entities can be nested, the property name is actually a path into the entity, e.g. `computer/screens/0/width`.

Entities are hosted in either a server or client *container*. The entities in a server container are made available for discovery and update by clients. The entities in a client container appear asynchronously as they are discovered after issuing a query (i.e. a *find* message) for matching entities to any server containers. Once created, both kinds of container are used in the same way: as a dynamic set of entities.

The Java code below defines the simplest useful entity in a Livespace, the shared clipboard.[3]

```
public class Clipboard extends Entity
{
  public String text;
}
```

The clipboard entity is as simple as it gets: it has one field, the shared text. However, the Entity base class, from which all entities inherit, adds two more fields: a globally-unique identifier (GUID) for the entity, and a name field used for presentation purposes. The Clipboard class being defined here is a programming convenience only, clients that replicate an entity do not need to have a class locally defined for it – if a class for an entity type is not available, a generic entity instance is generated instead.[4]

[3] Although the the Livespace Bus development API is currently implemented in Java, the communication protocol is platform-agnostic, meaning other languages can be supported (a C++ API is currently in development).

[4] We do not define formal schemas for entities on the bus beyond the kind of convenience class illustrated here. We have resisted the natural urge as computer scientists to define and enforce a formal schema system in order to avoid the friction that this would add in a distributed, changing and extensible environment. We have found in practice that the live schemas viewable by browsing entities in the Service Browser are sufficient – for developers, the key properties of all the core entities are defined and documented as Java classes.

The Java code below shows the process of publishing the clipboard:

```
// the connection to Elvin and the entity representing the room
Elvin elvin = ...;
Room room = ...;

// create entity server container
EntityServer server = new EntityServer (elvin, room, "clipboard");

// create clipboard instance
Clipboard clipboard = new ClipboardEntity ();

// publish it
server.entities.add (clipboard);
```

Once this code has been executed, the clipboard entity will be available on the bus for discovery by client containers. The client code below listens for changes to the clipboard and updates its text:

```
// listen for changes and print the old and new values
clipboard.addPropertyListener (new PropertyListener ()
{
  public void propertyValueChanged (PropertyEvent e)
  {
    print ("Clipboard text changed");

    print ("Old text: " + e.oldValue);
    print ("New text: " + e.newValue);
  }
});

// change the clipboard text
clipboard.setValue ("text", "Livespace clipboard example text");
```

While the clipboard is just a passive data holder, entities such as lights and projector displays that represent devices and services in the room will have an associated software 'driver' that is listening for changes to its entities, and which will synchronise those changes with the real world. So, when the Livespace projector service sees the powerOn property of the Display entity that it published change from 'false' to 'true', it sends the correct control code to the corresponding projector device to make it turn on.

The process of developing a Livespace service involves deciding on how best to model it as an entity, defining how changes to the entity map to the intended actions, and implementing listeners for the relevant properties which effect those changes. In the next major section you can see how some the of the core Livespace Bus services are designed in this manner.

The next two sections discuss some of the philosophy behind the entity replication approach and compare the Livespace Bus to some other related technologies

## 3.6. Find, Monitor, Update, Delete

Operations as varied as sharing text between applications, controlling room lighting, and switching video in a Livespace are all accomplished with just four verbs applied to one noun: *find* an entity, *monitor* changes to an entity, *update* an entity's state, and *delete* an entity.

This will be a familiar pattern to those from the database world, which is driven by Create, Retrieve, Update and Delete (CRUD), or those from the tuple space world of *get*, *put*, and *take*.[5] There are also parallels to be drawn with the Representational State Transfer (REST) approach to web services (Fielding 2000; Fielding *et al.* 2002), which is oriented around resources to which you may apply the HTTP operations PUT (create), GET (retrieve), POST (update) and DELETE (remove).

The advantage resource-oriented approaches is that they provide a uniform interface to a potentially complex space of operations: anything that can be done in these systems will be modelled as a state change to a resource, it's just a matter of deciding *what* to do with *which* resource. This is in contrast to the other widely used approach to distributed programming where a command verb is defined for every possible scenario. This works for small numbers of operations, but leads to a combinatorial explosion of verbs as the system grows. For example, for the clipboard, lighting and video switching examples in the previous section, we'd need to provide at least these verbs: *Find-Clipboard*, *Copy-Text-To-Clipboard*, *Get-Text-From-Clipboard*, *Get-Lights*, *Set-Light-Level*, *Get-Light-Level*, *Listen-To-Light-Level-Change*, *Find-Displays, Set-Screen-Switching*, and so on.

The approach where a command verb is defined for every possible action is embodied by many systems using the Remote Procedure Call (RPC) paradigm, such OMG's CORBA, Microsoft's DCOM or, more recently, Web Services based on SOAP (Gudgin *et al.* 2003). Aside from the above-described complexity engendered by this approach, the notion that procedure calls, object-oriented or otherwise, should be applied to distributed systems has been brought into serious question over the years, and many have come to the conclusion that distributed computing is such a fundamentally different problem from local computing that an approach that tries to simply extend paradigms from local computing is doomed to failure (Tanenbaum *et al.* 1988; Waldo *et al.* 1994).

## 3.7. The Livespace Bus Compared To Other Technologies

A number of the key architectural concepts underlying the Livespace Bus were inspired by a existing technologies. These include systems based on tuple spaces such as iROS (Johanson *et al.* 2001) and Jini, as well as the data access abstraction technology known as Service Data Objects (SDO) (Beatty *et al.* 2003).

The key difference between the Livespace Bus and tuple space systems is its resource-oriented nature. At first glance you might assume that the tuples used by systems like Jini and iROS are the equivalent of a Livespace entity: however the tuples in these systems are usually employed as events for discovering services and invoking operations on them, they do not model anything in themselves. These systems are architected around services and the operations they provide, rather than around resources and the state

[5] Examples of systems based on tuple spaces include Linda (Gelernter 1985) or Jini (Waldo 1999).

changes they support.[6] And while, in practice, many of these operations end up being about monitoring and updating state,[7] the system itself does not explicitly model the concept of a stateful resource.

The SDO framework also has some features in common with the Livespace Bus. SDO is oriented around data access using graphs of abstract data objects, which have much in common with Livespace entities. SDO clients may also employ an optimistic update strategy, where the client modifies a local copy of a data object graph and transfers a change summary to the data store at a later time. However, similarities to the Livespace Bus end at this point, since SDO was primarily designed to be a data abstraction layer for accessing persistent stores such as relational databases or XML documents. SDO does not address the issue of object discovery or support broadcasting updates to rapidly-changing objects. It also does not define a model for monitoring changes to objects.

[6] In many ways these sorts of service are the equivalent of objects in an object-oriented system: they intentionally hide their state, encapsulating it behind the operations that manipulate it.

[7] This especially true of iROS, which deals with a similar set of abstractions to a Livespace.

# 4. Livespace Core Services

In this section we run through the Livespace services that collectively form the core of a basic Livespace. These are mostly entity-based services, however a few lower-level services, such as the Livepoint remote mouse control service, operate by sending and receiving messages directly via the Elvin router. The choice to use the router directly may be made either for performance reasons, or because the service does not map naturally to the entity-modelling architecture.

## 4.1. Room Management Services

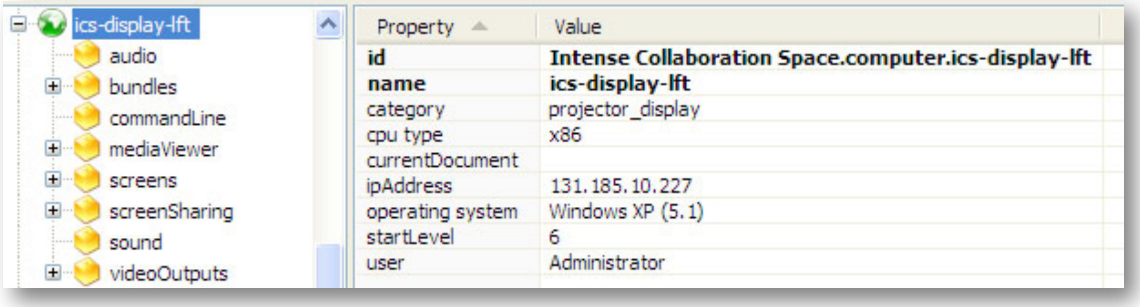
The room management services provide the core services supporting the operation of the room's computers and devices, as well as basic facilities, such as media playback, that are built on by higher-level applications.

### The Room Service

The room service publishes an entity that models the Livespace room itself, providing essential metadata used by other services. It also acts as a namespace for its entities so that, in the case where multiple rooms are connected, it is possible to issue a query for all entities, or restrict the query to just the entities in a given room.[8] In the Service Browser application, the room is presented as the root of the tree to visually indicate this – although the other entities in the space are not actually children of the room entity, they exist within its namespace.

### The Computer Service

Each computer in a Livespace advertises its presence by running a service which publishes a single descriptive computer entity. Below is a screenshot from the Service Browser application showing some of the top-level properties of a computer entity.



| Property | Value |
|---|---|
| **id** | **Intense Collaboration Space.computer.ics-display-lft** |
| **name** | **ics-display-lft** |
| category | projector_display |
| cpu type | x86 |
| currentDocument | |
| ipAddress | 131.185.10.227 |
| operating system | Windows XP (5.1) |
| startLevel | 6 |
| user | Administrator |

*Part of a computer entity*

[8] We would like to investigate using a more scalable approach to namespaces in a future version of the Livespace Bus, which would employ a DNS domain-style namespace hierarchy for entities, allowing one to query with varying levels of specificity: for example 'find all entities in the *defence.gov.au* namespace'.

The computer service is one of the core services in a room since, as well as advertising the existence of a computer and basic information about its hardware, it also serves as a substrate for other services to attach to and extend. Many services that are logically associated with a computer add extended attributes to the computer entity instead of publishing a top-level entities of their own.[9] This approach helps to reduce the proliferation of tiny, top-level entities that would otherwise occur, and makes it possible to see all computer-related properties in one place. Several examples of extension services appear below – other extension services include one that looks up video switch ports for the video inputs and outputs advertised on computers and displays.

### The Media Service

The media service extends the computer entity to add the ability to play back audio/visual files, display and control Microsoft PowerPoint presentations, and open documents and web pages in a browser.

To open a file on a computer, a client puts the link to the file (a URI) in the computer's `currentDocument` property, which will cause the file to be opened immediately in a new application or web browser window. Playing audio/visual media files[10] or opening a slide presentation can be done in the same way or, if the client wishes more control over placement and playback, it can instead put the link into one of the sub-portals inside the computer's `mediaViewer` property, and set the `state` property of the portal to 'playing'.[11]

As well as providing a generalised, distributed multimedia playback facility for the room, the media viewer can also be scripted to generate automated presentations by the Meta Applications service (see the explanation of Application Services later in this document).

### OSGi Service Administration

The OSGi administration service, somewhat self-referentially, is a service that manages services. It operates by publishing a model of a computer's services in the `bundles` property of the `computer` entity. As well as showing the services on a computer and their state, this model can be manipulated to remotely add, remove, start, and stop services.

The application that is generally used to display and manage services using this capability is shown in the description of the OSGi Administration application described in a later section.

[9] These extension services that attach, or 'glom onto', other entities have also become informally known as 'glommer services'.

[10] The media service is implemented using Apple's QuickTime multimedia framework, enabling the wide variety of media supported by the QuickTime platform to be played.

[11] These portals define sub-regions of a screen in order to manage playback of multiple audio/video streams or PowerPoint presentations on a single screen.

### Screen Sharing

The screen sharing service allows remote computer displays to viewed over a network. It is often used for pushing the display of a personal computer up to a shared display, or for sharing one computer's display with several people to make collaborative use of an application. Under the hood, the screen sharing service uses Virtual Network Computing (VNC) technology – an alternative service allows the use of a hardware video switch if available.

The section on the Screen Sharing application later in this document has a fuller description of what this service provides.

### Lighting

The lighting service acts as a front end to an automated lighting controller, publishing a set of light entities to match those in the room and synchronising them with the underlying hardware. Predefined lighting scenes can then be defined as part of the Sessions service.

There are a number of off-the-shelf options that provide automated lighting. DSTO's ICS uses the Clipsal C-Bus system (C-Bus 2007). Other similar systems include consumer home-automation products based on standards such as X10 (X10 2007).

### Displays

Each stand-alone display in the room, such as a projector, is represented by a `display` entity. A display entity may be merely a passive entity representing the existence and capabilities of a display, or it may be managed by a driver service that controls the display hardware, for example an LCD projector controlled via an RS-232 serial interface.

For displays that can be targeted using a video switching device, a `videoSourceId` property on the display is used to select the current video source for the display, which is typically the output of a computer, camera, or VTC system. Displays that can can be turned on or off programmatically, such as LCD projectors, will also have a `powerOn` property.

### Audio Mixing

If the Livespace's audio is managed by a software-controllable mixer, the driver that controls the mixer may publish a set of `audio mixer` entities that allow basic control of the audio channels. For example, mixer entities such as 'Left', 'Right', 'Centre' and 'Master' may be available: each of these entities will have a `volume` property allowing the volume of that channel to be adjusted between 0 and 100.

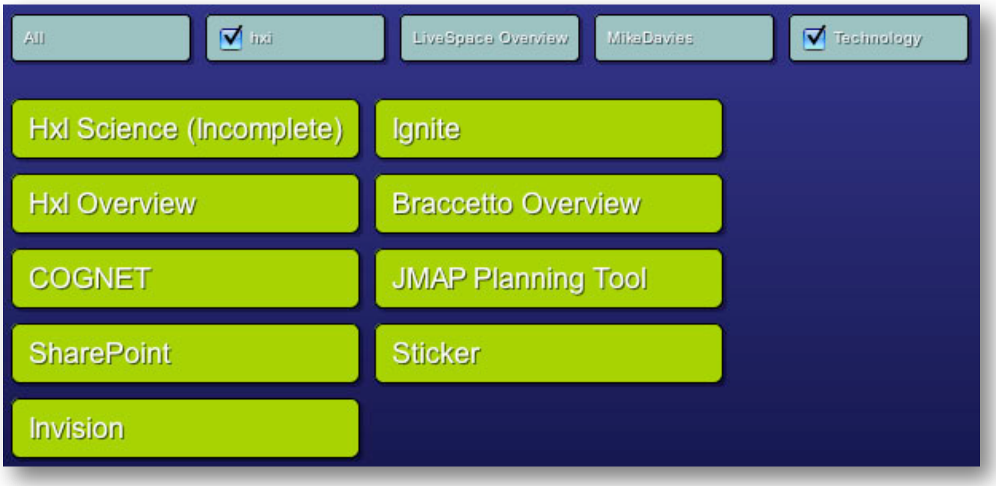
## 4.2. Application Services

The application services are a level of abstraction over the services described above. Application services provide the the basis for most of the user-visible applications of the room.

## Meta Applications

The meta applications service is so named because it provides a 'meta service', a service that coordinates other services. The *meta applications* stored and executed by this service can be thought of as scripts that direct a series of actions applied to other services. Ultimately these actions, like all Livespace operations, boil down to manipulating the state of entities – the meta applications service makes it possible to execute a series of synchronised, coordinated state changes across the services in a room.

Meta applications are often used in concert with the capabilities of the Media Service to generate automated multimedia presentations, in which audio/visual displays are coordinated across a number of screens. The description of the Meta Application Editor later in this document includes some examples of this kind of meta application.



*Some of the meta applications available in the ICS*

## Sessions

The sessions service provides the ability to quickly switch the room into pre-defined configurations, or *sessions*. A Livespace has at least two fundamental sessions: 'Room On' and 'Room Off' – these will typically be set up to turn the room's core devices on or off when users wish to enter or leave the room. For example, the 'Room On' session might activate the 'Presentation Lighting' scene, turn the room's main projectors on, forward default desktops to the projectors, and play a welcome sound, while 'Room Off' would essentially reverse those actions.

Custom sessions may be defined to switch the room into a state convenient for common tasks. For example, a 'Brainstorming' session might set up the room so that the screen of the computer at the head of the table is switched to the main projector (in order to allow the chair of the meeting to share their display), turn up the lights over the meeting table, and open TeamThink on all the computers on the table ready for recording ideas.

Under the hood, a session is actually just a type of meta application. The sessions service extends the meta application model by managing a stack of session meta applications, enabling a session to be pushed on to the top of the stack to become current, and to be popped off the top of the stack to roll back to the previous session.

## Clipboard

The clipboard service, which has already been described in the Livespace overview sections, provides a way to share text across computers. Common uses include:

- Pasting links from a web browser for other people to open,
- Transferring text that needs to be conveyed without errors, such as codes, web addresses, file paths, or formal instructions, and
- Copying and pasting sections of text from documents or web pages to cite during a discussion.

## Presence

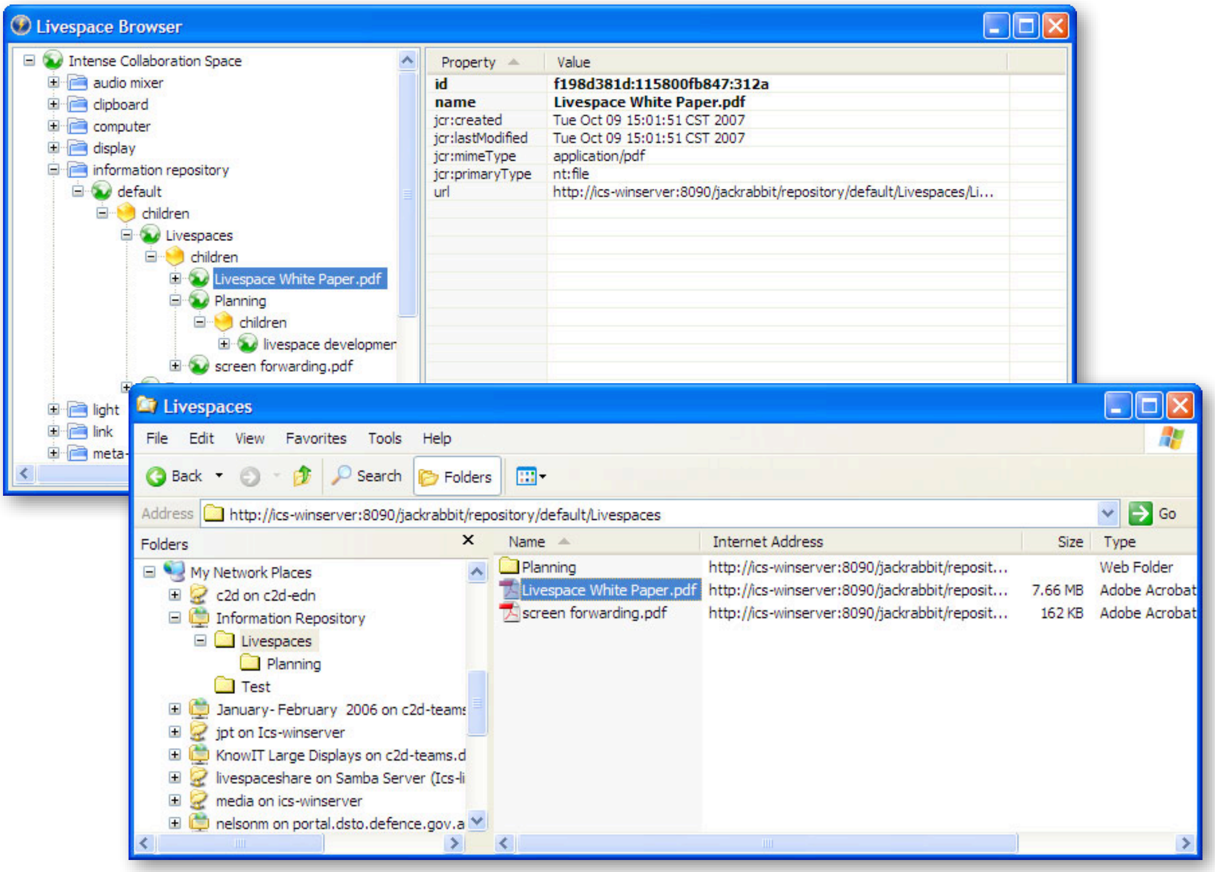
The presence service publishes information about who is in the room, also known as *virtual presence*. The published information currently consists only of the user name the person used to log in to their computer. In future we anticipate adding information such as whether the person appears to be currently using the computer based on keyboard and mouse activity, and other information including email addresses and, in a military context, rank and clearance level.

The presence information is also extended by other services. For example, TeamThink extends its user's presence information to include the TeamThink-specific fields that advertise what role the person is playing and the text the person is currently typing.

## Information Repository

The information repository provides storage for documents, media and other information resources. In some ways the repository is analogous to a shared network drive, however the repository provides a number of capabilities beyond that of a traditional file system, including change notification, full text indexing and file versioning.

The repository appears as a tree-structured entity in the Livespace, which can be accessed in the same way as other entity-based services. It can also be accessed like a normal file system from desktop applications like Windows Explorer and Mac OS X Finder via a WebDAV interface.



| Property | Value |
|---|---|
| **id** | **f198d381d:115800fb847:312a** |
| **name** | **Livespace White Paper.pdf** |
| jcr:created | Tue Oct 09 15:01:51 CST 2007 |
| jcr:lastModified | Tue Oct 09 15:01:51 CST 2007 |
| jcr:mimeType | application/pdf |
| jcr:primaryType | nt:file |
| url | http://ics-winserver:8090/jackrabbit/repository/default/Livespaces/Li... |

Address http://ics-winserver:8090/jackrabbit/repository/default/Livespaces

| Name | Internet Address | Size | Type |
|---|---|---|---|
| Planning | http://ics-winserver:8090/jackrabbit/reposit... | | Web Folder |
| Livespace White Paper.pdf | http://ics-winserver:8090/jackrabbit/reposit... | 7.66 MB | Adobe Acrobat |
| screen forwarding.pdf | http://ics-winserver:8090/jackrabbit/reposit... | 162 KB | Adobe Acrobat |

*The information repository in the Service Browser and in Windows Explorer via WebDAV*

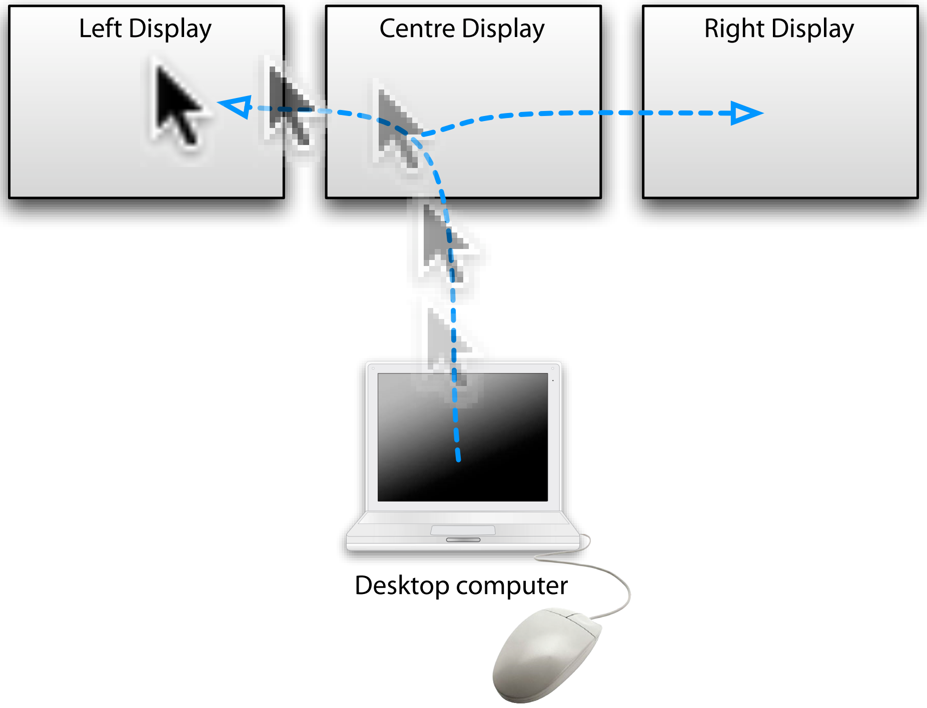
Representing the repository as as an entity allows Livespace clients to easily discover, access and monitor it. In order to add new content, a client need simply create a new entity as a child of a folder node in the repository. The repository service will then automatically create a new resource in its underlying store and add the URI generated for the resource as a property of the entity – the client can then access the resource via HTTP GET (read) and PUT (write) operations. We represent the resource data as a URI, and not directly as a property of the entity, because the data may be large and HTTP resources are more readily accessible by existing desktop applications.

At the time of writing, a prototype of the Information Repository is being trialled. The service that supports it is based on the Apache Jackrabbit (Apache 2007) implementation of the Java Content Repository specification (JCR 2007). The hybrid of Jackrabbit and WebDAV as the back end of the service, with the entity-based front end, is so far proving very effective in day to day use.

## Livepoint

Livepoint provides the ability to remotely control a screen by allowing the mouse to virtually jump from the user's local PC to the remote screen. While the jump is in progress, the user's mouse and keyboard act exactly as if they are plugged into the remote computer.

Typically one or more edges of a user's screen are designated as Livepoint jump points. For example, in the ICS Livepoint is configured such that pushing the mouse to the top edge of the screen on one of the meeting table computers causes control to jump to the centre shared display at the front of the room. Moving the mouse to the left edge of the centre display jumps the mouse to the left display, and the same for the right display. This allows people to naturally shift control between their local PC and the three computers that drive the shared displays, and is remarkably effective in making them feel like one large desktop space.



*Using Livepoint to jump between screens*

Livepoint builds on the core capability provided by the Synergy application (Synergy 2007).[12] Synergy provides the platform-specific logic for capturing and injecting keyboard and mouse events that is needed to simulate one computer's keyboard and mouse being connected to another computer. Livepoint removes the client/server distinction employed by Synergy and instead enables any computer in the room to send or receive keyboard and mouse events via the Elvin event router. It also extends Synergy to be aware of the room's current screen forwarding state.

[12] Livepoint also derives inspiration from iROS's PointRight application (Johanson, *et al* 2002).

# 5. Deployment

A Livespace system consists of 60 or more software components, deployed onto a number of computers: typically 10 or more for a reasonably well-equipped room. Different sets of components are needed on computers with different roles: the software deployed to a server computer will be quite different than that on one of the desktop computers. The challenge is to manage the configuration and deployment of the software into the room so that it is possible to reliably add, remove, update, and change parameters of the software and hardware.

To address this, we have built a system based on the OSGi service framework (OSGi 2007). OSGi is an open standard service platform, originally developed for the mobile arena. The OSGi framework provides the following key facilities:[13]

> **A software component model**. OSGi enables packaging software into components called *bundles*. A bundle contains the executable code for a component as well as metadata, such as the component's name, version and, most importantly, its dependencies on other components.
>
> **A managed service runtime environment**. OSGi bundles are deployed and executed inside an OSGi service manager that hosts bundles by first resolving their dependencies on other bundles, linking them together, and then running the code. Bundled services can be deployed, undeployed, started, and stopped dynamically, without affecting other services in the environment.

We have built a system on top of OSGi to support our own specialised requirements. In particular we have added the following three facilities.

## 5.1. The Client Shell

The client shell is a tiny (4 kilobyte) program that connects to the room's main server via HTTP and starts the Livespace software. It first copies and executes a single 'bootstrap' OSGi bundle provided by the server, which then identifies and starts the rest of the bundles required for the host: these are also copied from the room server.

This approach means computers in a Livespace only require a one-off install of the client shell software: further software upgrades are carried out by simply updating the bundles on the server, which will be pushed out to clients next time they restart. The client shell can be remotely restarted using the management system if required.

The client shell can optionally be installed as a system service, meaning it is always running even when no one is logged into the computer. While this can be useful, especially on server computers, there are a number of security issues to take into account when running bundles with system privileges if they need

[13] At the time of writing there are at least three open source OSGi implementations available: Knopflerfish (Knopflerfish 2007), Apache Felix (previously known as Oscar) and Equinox (which provides the core of Eclipse's plug-in architecture). We are currently using Knopflerfish.

to interact with the desktop.[14] We have investigated several ways to work around this, but so far the only feasible solution is to simply run the the client when the user logs on rather than as a service.

## 5.2. Configuration Management

This Livespace central configuration system determines both *deployment* (what bundles are deployed to which hosts), and *settings* (the configuration parameters for each bundle). Both of these types of configuration can be targeted in the following ways: they may be applied globally to all hosts, targeted at a specific category of computer (for example, 'desktop computers on the table run Livepoint, server computers do not'), or targeted at a specific host (e.g. 'Ignite on the touch-screen panel host always runs on startup in full screen mode').

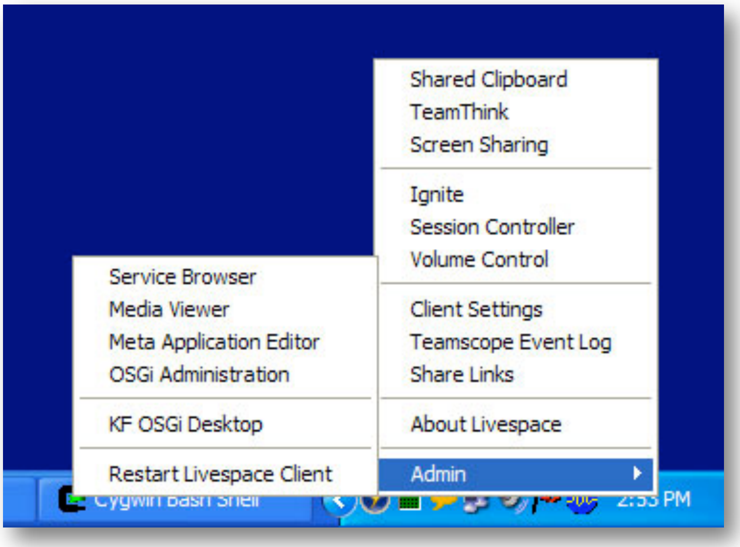
The configuration for all hosts in a room is stored on the computer acting as the main server for the room: the aggregated set of deployment and configuration parameters on this server can be thought of as the room's configuration. Since the configuration simply consists of human-readable text files, it can be managed with a conventional software configuration management (SCM) system.

A configuration is a split into two layers: the default layer that we ship with the Livespace distribution, which defines the standard set of configuration parameters; and a room-specific layer that contains the overrides and additions needed to set up a specific room. The 'room' layer is logically merged on top of the default layer to generate the actual configuration, an arrangement which allows us to manage changes to the configuration by simply modifying the default layer in new releases. We also provide a configuration template with embedded instructions that is designed to make initial setup easier.

## 5.3. The Desktop Dashboard

The desktop application components in a Livespace, such as Clipboard, TeamThink, Ignite, etc., are themselves OSGi bundles. On computers that interact with the room's users we deploy a bundle named the *Dashboard* that acts as a launcher for these desktop application bundles. The dashboard appears as an icon in the Windows or Linux task bar, or in the application menu on the Mac: clicking the icon shows a menu of applications that can be launched.

[14] These security issues are severe enough that some operating systems such as Mac OS X and Windows Vista simply do not allow system services to interact with the desktop at all.



*The Dashboard menu in the Windows notification area*

The items shown on the dashboard are controlled using the Livespace configuration system, so they can be customised for the room. They can also be customised on a per-computer basis, allowing different application sets for different categories of computer.

# 6. Federation

The Livespaces system was designed from inception to support multiple connected rooms, and to support connection of the services in those rooms. To this end, we have developed a federation service, which is an optional component that allows two or more rooms to be connected together, either permanently, or on an *ad hoc* basis.

As an example of what we mean by federation, consider the simplest scenario, where we have two Livespaces in an single organisation that we wish to connect for a meeting.[15] In this scenario, we might choose to set up federation to do the following:

- Synchronise the clipboards of the two rooms so that pasted text in one room appears in the clipboard of the other room.
- Connect the screen sharing application so that screens in one room can be shown on screens in the other.
- Synchronise the meta applications between the rooms, so that an automated presentation played in one room also plays in the other.
- Synchronise presence so that applications such TeamThink, which rely on virtual presence, can see all people in both rooms.

This is a good setup for two rooms in one organisation, but the federation setup would probably be quite different in other scenarios. For example, you may not wish to synchronise the meta applications in the general case. And in a situation where sensitive information may be in use, you may not wish to allow screen sharing. For this reason, the federation system allows a great deal of control over what resources are connected.

Internally, the Livespace federation service works by connecting entities on the Livespace Bus in one of two ways:

> **Replication**. In this mode, an entity in one room is synchronised with another: the other entity becomes an exact replica of the primary entity. For example, this mode of federation is used to synchronise the clipboards of two rooms.
>
> **Mirroring**. In this mode, a selected set of entities in one room are 'reflected' into the other. From the point of view of the other room, those entities appear as normal entities: the federator ensures that changes in either room are reflected in the other. This mode of federation is used for example, to reflect virtual presence information from one room into another.

The federation system can be thought of a replication proxy between the rooms: as such it can also provide the extra management needed when proxying connections over a Wide Area Network (WAN). The communication characteristics of a WAN can be very different from the Local Area Network (LAN) environment on which the Livespace Bus is generally used: a WAN will often have far higher communications latency, variable bandwidth, lower reliability and lower security than a LAN

[15] Federation across organisational or trust boundaries is complicated by the extra security requirements implied. Although the Livespace federation system has a number of options to control what goes across federation barriers, for simplicity we won't use them in this example.

environment. The built-in defaults of the Livespace Bus, including critical parameters such as response time-outs, are optimised for a LAN environment: the federation service, as it proxies between buses, can adapt this to behaviour better suited to higher-latency links. Its nature as an external, optional service also de-couples the federated rooms, shielding them so that if the network between them goes down, their local services remain unaffected.

# 7. Applications

This section provides an overview of the applications that are presented to users of a Livespace. These applications are typically launched from the Livespace Dashboard menu.

## 7.1. Clipboard

The Livespace clipboard has already been covered previously. It provides a simple mechanism for quickly transferring text and URIs between participants.

## 7.2. TeamThink

TeamThink is a collaborative document editor. It is a simple tool that is most effective for brainstorming or other collaborative authoring sessions between two to four people, where each person has a task that can be done in parallel to the others. During a TeamThink session, the meeting convenor will periodically synchronise with the others by stopping them and pulling their inputs into a consolidated document.

A TeamThink session will generally consist of one or more cycles of the following steps:

**Decide roles**. Typically the meeting convenor is the Editor, others are Proposers.

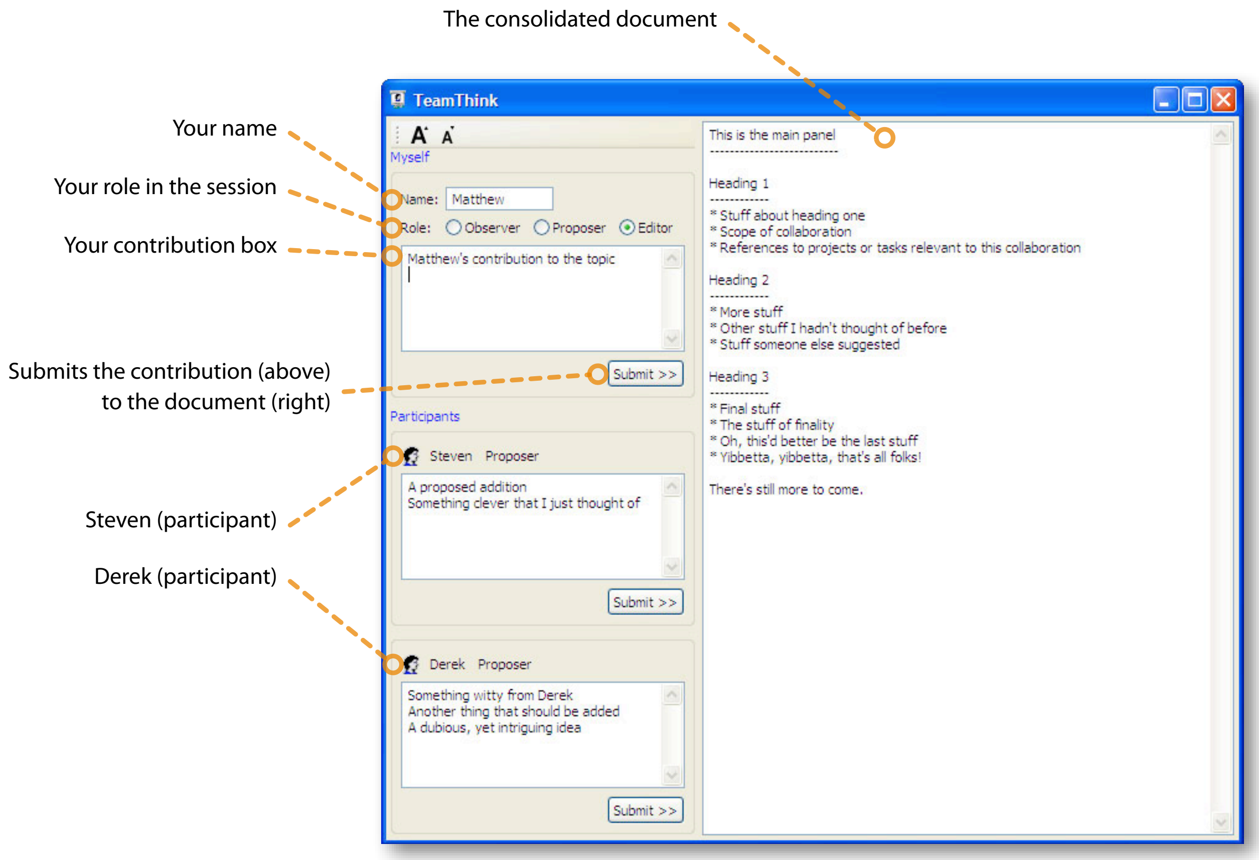
**Assign tasks**. For example, the editor may ask each person to generate a list of key points for a different section of a document.

**Work in parallel**. Each person enters their contribution in their text area. The Editor may contribute also, or simply monitor other people's progress.

**Synchronise and consolidate**. The editor calls a halt and interactively runs through each person's contributions, possibly modifying them. The editor then submits the text contributed by each person into the appropriate part of the main document.

The end result is a consolidated plain text document which may then become the base of a more formal document, presentation, email, etc.

The TeamThink window below is showing a session with three participants. Each person is running TeamThink on their computer and can see the others in the vertical list on the left. Each person has a role and a personal area that they can enter text into at the top of the left hand panel. There is generally one person in the Editor role who is responsible for directing the meeting, with the others being Proposers (people can also opt to run in Observer role to watch what's going on).



The consolidated document

Your name

Your role in the session

Your contribution box

Submits the contribution (above) to the document (right)

Steven (participant)

Derek (participant)

Name: Matthew

Role: ○ Observer ○ Proposer ● Editor

Matthew's contribution to the topic

Steven Proposer

A proposed addition
Something clever that I just thought of

Derek Proposer

Something witty from Derek
Another thing that should be added
A dubious, yet intriguing idea

```
This is the main panel
-------------------------

Heading 1
------------
* Stuff about heading one
* Scope of collaboration
* References to projects or tasks relevant to this collaboration

Heading 2
------------
* More stuff
* Other stuff I hadn't thought of before
* Stuff someone else suggested

Heading 3
------------
* Final stuff
* The stuff of finality
* Oh, this'd better be the last stuff
* Yibbetta, yibbetta, that's all folks!

There's still more to come.
```

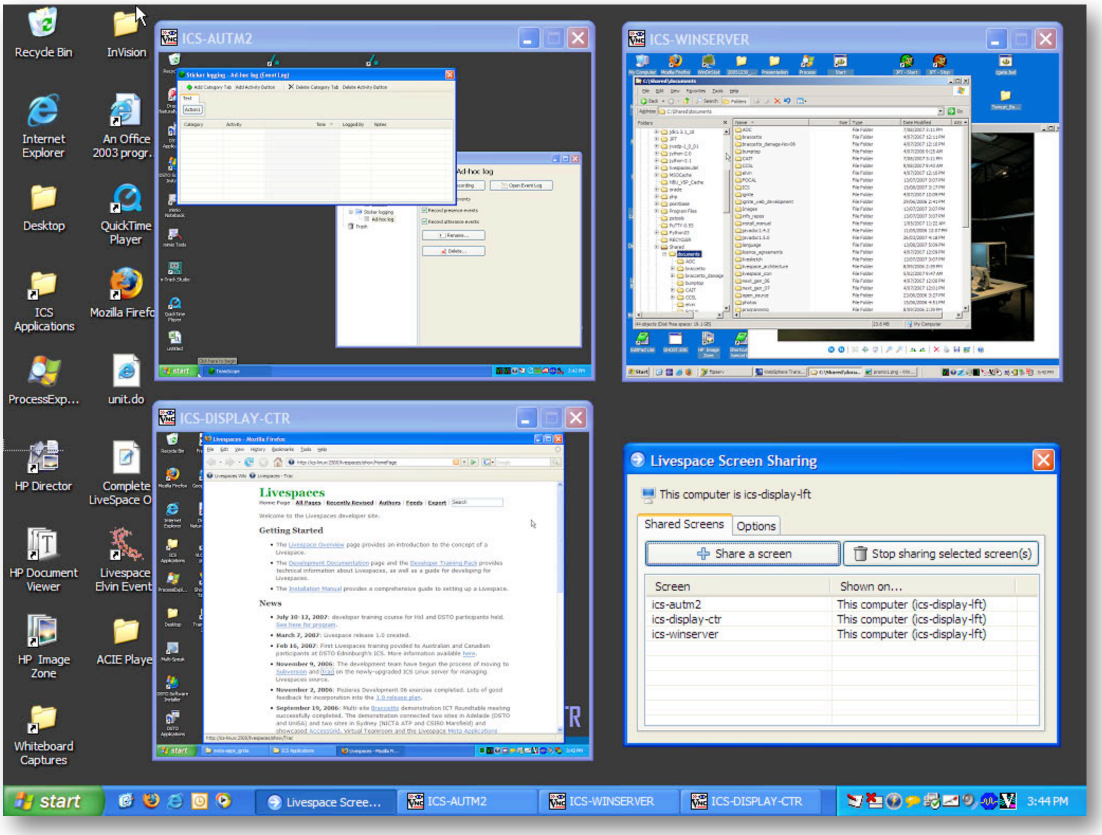
*The TeamThink window*

All of the information in the window is live: as people edit it will be updated on everyone's screen in real time. The Editor is the only one who can edit the main document, and can even edit other people's text if they want, although this is usually done during the discussion and consolidation phase, to propose a change before submitting, for example.

TeamThink is a model example of how the Livespaces infrastructure can accelerate prototyping new collaboration applications. The TeamThink concept actually came directly from one of our military users just before an experimental exercise that was to be hosted in a Livespace environment – we saw an opportunity and the first prototype was implemented in under two days and successfully trialled during the exercise. TeamThink today still only weighs in at a tiny 700 functional lines of Java code. This type of rapid development and feedback would not have been possible without the running start provided by the Livespaces infrastructure.

## 7.3. Screen Sharing

The Screen Sharing application allows the screen of a computer to be viewed live on another. The screen can be sent either in display-only mode, or in fully interactive mode, where the user viewing the screen can control the mouse and keyboard. The screen display can also be scaled down or up, allowing multiple screens to be viewed at once, as shown below.



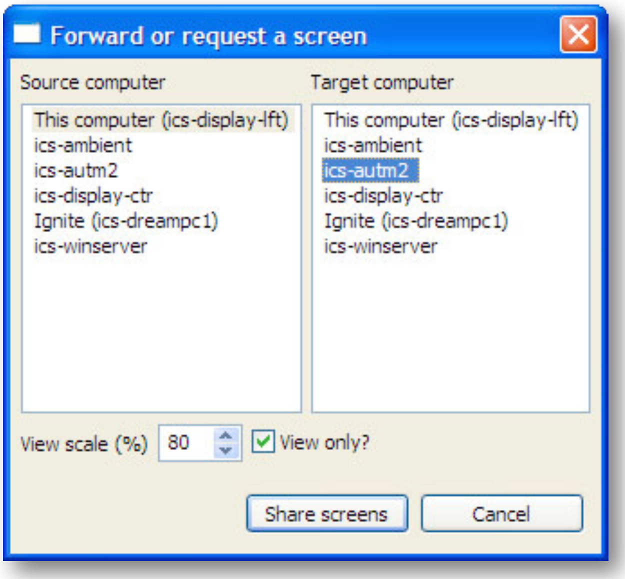
Livespace Screen Sharing

This computer is ics-display-lft

| Screen | Shown on... |
|---|---|
| ics-autm2 | This computer (ics-display-lft) |
| ics-display-ctr | This computer (ics-display-lft) |
| ics-winserver | This computer (ics-display-lft) |

*Screen sharing multiple screens to one display*

To use the screen sharing application, the user activates it from the Dashboard menu, which displays the window shown in the bottom right of the screenshot above. From this window the user can see what screens are already shared, create new shared screens, and un-share existing ones. The 'Share a screen' button displays a list of source and target computers to share from and to. For example, the window below shows the user about to send their screen ('ics-display-lft') to a shared display computer ( 'ics-autm2').



Source computer

This computer (ics-display-lft)
ics-ambient
ics-autm2
ics-display-ctr
Ignite (ics-dreampc1)
ics-winserver

Target computer

This computer (ics-display-lft)
ics-ambient
ics-autm2
ics-display-ctr
Ignite (ics-dreampc1)
ics-winserver

View scale (%) 80

☑ View only?

*The screen sharing window*

The screen sharing application uses Virtual Network Computing (VNC) technology to perform the sharing over the network.

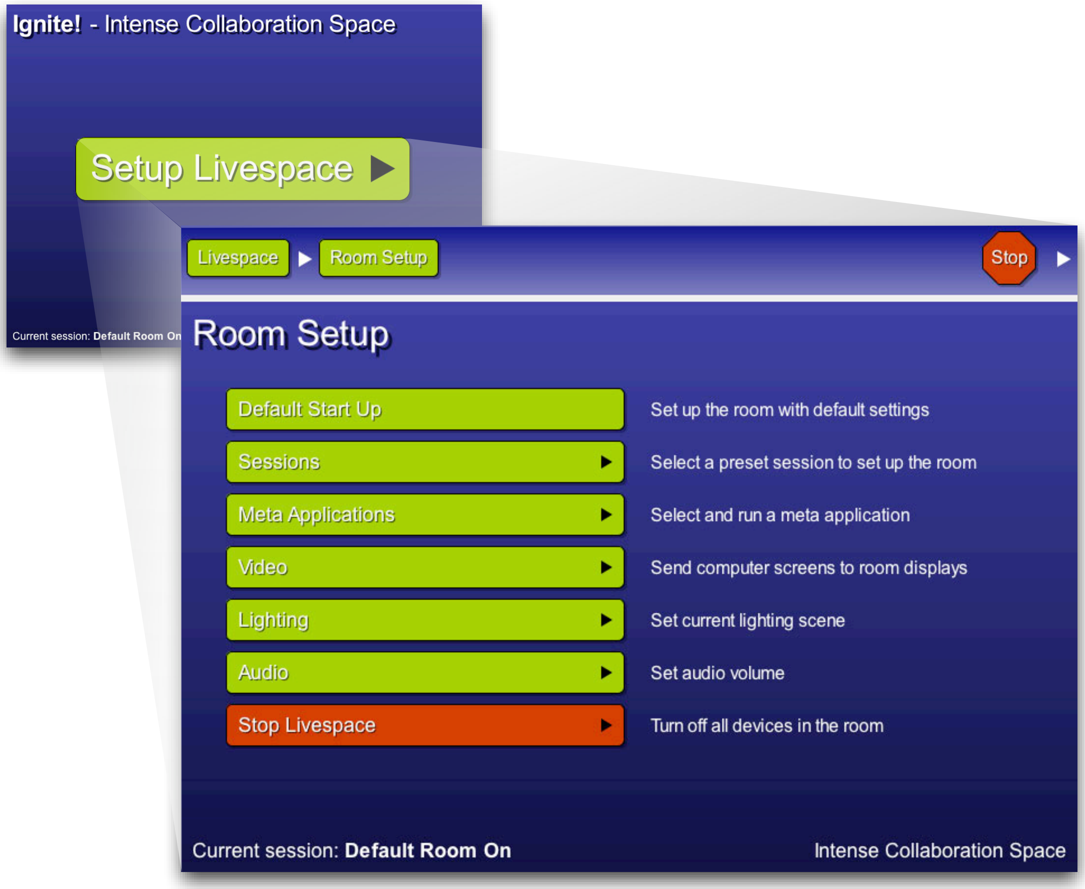
The current implementation of the screen sharing application is a prototype designed to enable experimentation with the best ways to use it and the policies that are needed to control it. There are clearly important awareness and privacy issues to consider when deploying such a powerful way of monitoring any computer in a Livespace. One of the keys issues to be addressed is the how to make the people and services in the room aware of the visibility of any given screen. For example, if you're not aware that someone in a remote room has forwarded your screen to one of their shared displays, you may be embarrassed when writing an in-confidence text message to someone at the other end.[16] Similarly, the instant messaging client on the other end would ideally not flash such a personal message on the screen when the display it is on is publicly visible.

## 7.4. Ignite

Ignite is the primary control panel for the room, and is the interface most users will see when they first enter a Livespace. As such, Ignite is intended to provide an approachable interface for controlling the room's lights, displays and other devices, switching video and audio, and playing meta applications.

Most Livespaces feature a touch-panel mounted on a wall at the entrance to the room running Ignite in full-screen mode. Because of this, the Ignite the interface is designed to make it easily controlled via finger taps on a touch panel, although it can just as well be used on a desktop with a standard mouse.

[16] The current version displays an icon in the computer's notification area when the screen is being shared to another computer. The details of where the screen is being shown are available as a tooltip on the icon.



Ignite! - Intense Collaboration Space

Setup Livespace ▶

Current session: Default Room On

Livespace ▶ Room Setup

Stop ▶

Room Setup

| | |
|---|---|
| Default Start Up | Set up the room with default settings |
| Sessions ▶ | Select a preset session to set up the room |
| Meta Applications ▶ | Select and run a meta application |
| Video ▶ | Send computer screens to room displays |
| Lighting ▶ | Set current lighting scene |
| Audio ▶ | Set audio volume |
| Stop Livespace ▶ | Turn off all devices in the room |

Current session: **Default Room On**

Intense Collaboration Space

*The initial Ignite start page leading on to Room Setup*

The screen shots above show the initial Ignite 'Setup Livespace' page leading to the 'Room Setup' page, which provides a central menu of options for controlling the room. From here the user can simply tap 'Default Start Up' to activate the default session, which will typically turn lights on, start projectors, select the default video and audio forwarding, etc. Alternatively, the user may select a custom session by tapping the 'Sessions' button, which takes them to the page described in the next section.

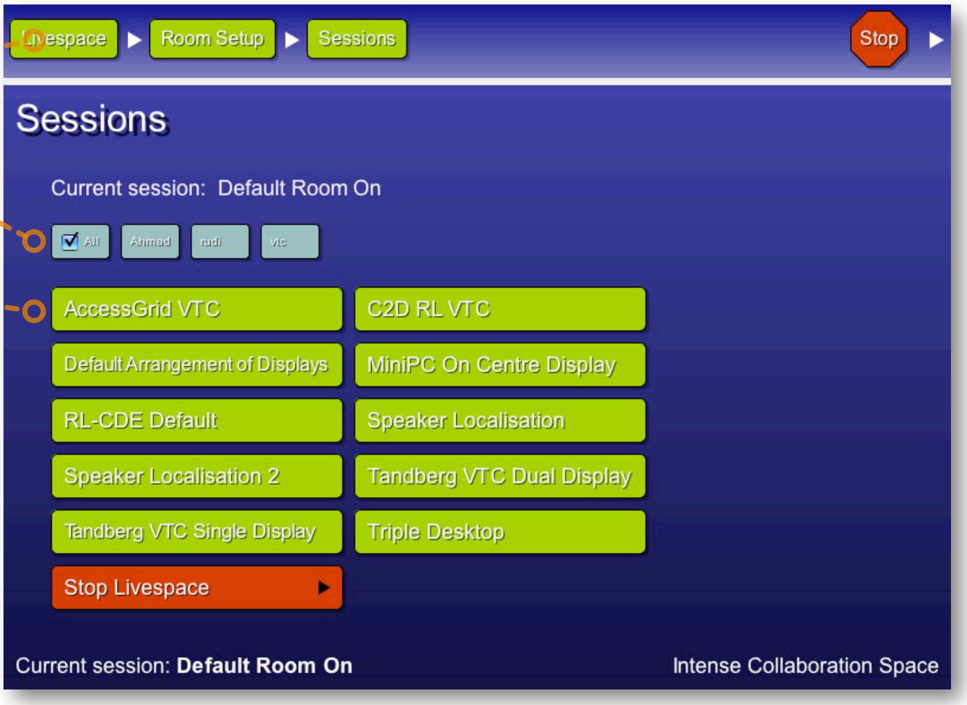
Having the 'Setup Livespace' page as the starting point, rather than going directly to the 'Room Setup' page, has been the subject of some debate within the development team. The idea is that the large, green 'Setup Livespace' button is easily recognisable from a distance: even users completely unfamiliar with Ignite should notice it and walk over, at which point there is only one option available ... tap the button. Successfully navigating to the setup page in this way should convey the idea that green buttons are tap-able actions, and that an arrow on a button means it takes you to another page.

## The Sessions Page

Breadcrumb trail showing navigation history

Filter session list to show only selected categories

Available sessions



Current session: Default Room On

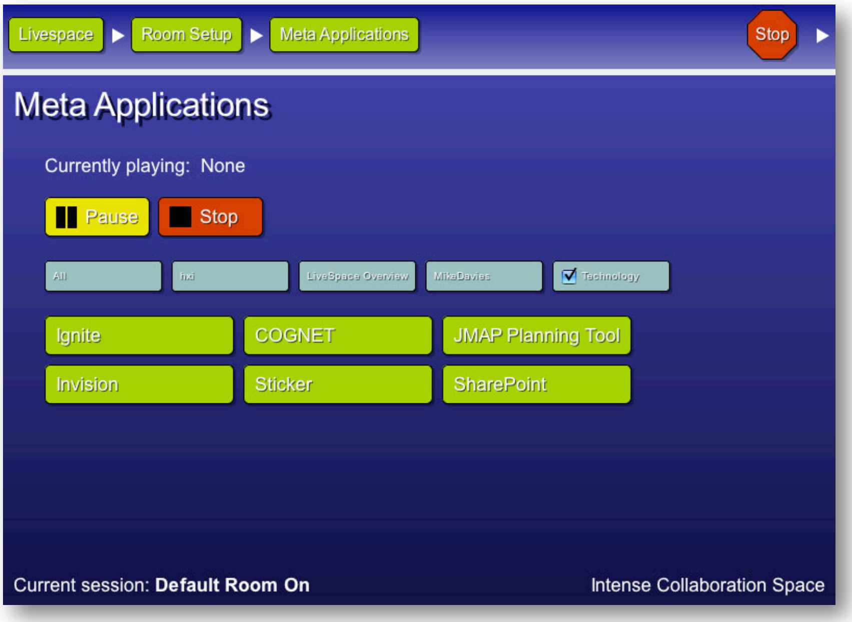
Current session: **Default Room On**

*The Ignite Sessions page*

The Sessions page shows the list of all available pre-defined sessions, or a filtered subset if the user selects one of the filter categories above the list. Tapping a session activates it – the current session is always shown at the bottom left of the Ignite display. For example, the user could tap the 'AccessGrid VTC' session to switch the computer that runs the AccessGrid video teleconferencing software onto the centre display, start the AccessGrid software, and switch lighting into the configuration best suited for a video teleconference.

Note also the 'breadcrumb' navigation trail at the top of the page – this allows you to see how you got to the page, and allows you to go back by tapping one of the buttons. The user may also shut down the room from any page by tapping the 'Stop' button that appears in the top right corner.

## The Meta Applications Page



Meta Applications

Currently playing: None

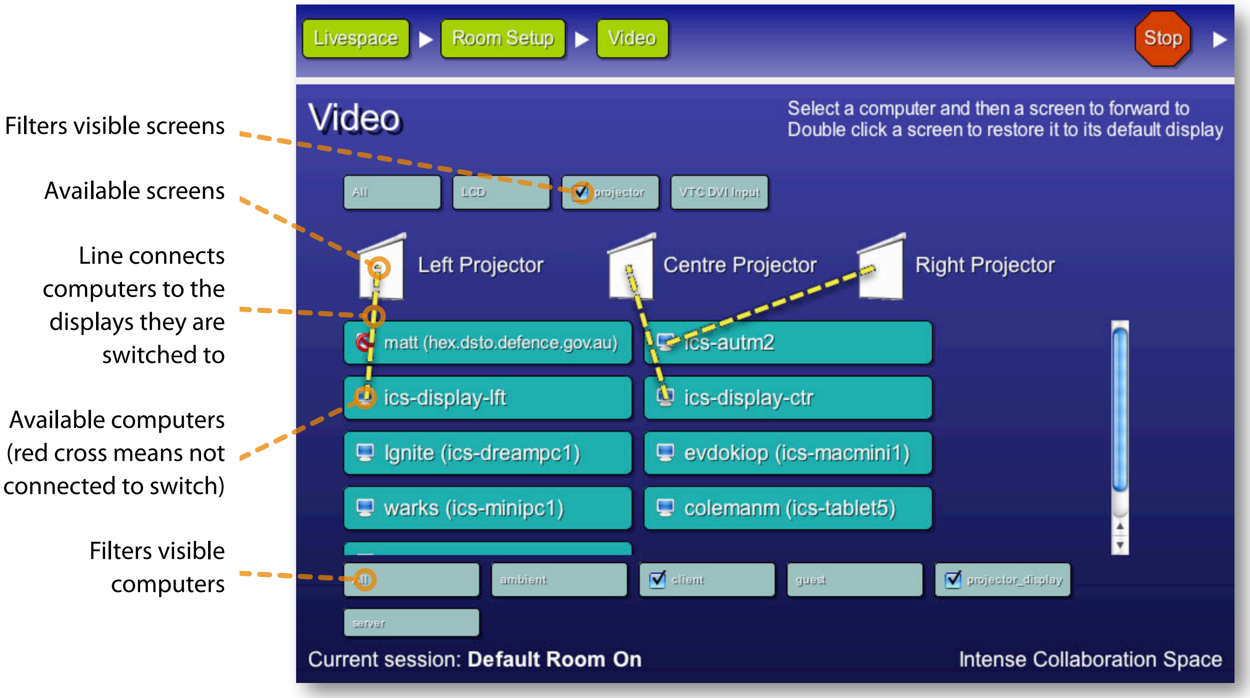
Current session: **Default Room On**

Intense Collaboration Space

*The Ignite meta applications page*

The Meta Applications page is similar to the sessions page, adding Play/Pause and Stop buttons above a list of pre-defined meta applications. The similarity is no coincidence since sessions are actually just a special type of meta application.

## The Video Page



Filters visible screens

Available screens

Line connects computers to the displays they are switched to

Available computers (red cross means not connected to switch)

Filters visible computers

*The Ignite video control page*

The video controller page is the user’s interface to the room’s video switching capability. The displays in the room that can be targeted by the room’s video switch are shown at the top, computers below, labelled by their current user (if any) and the computer’s name. Computers that cannot be forwarded because they are not connected to the switch, such as the laptop being used by ‘matt’ (top left computer in the figure), are shown with a red cross over their icon. The yellow dotted lines show the current switching state, for example the ‘Centre Projector’ display is currently showing the the ‘ics-display-ctr’ computer.

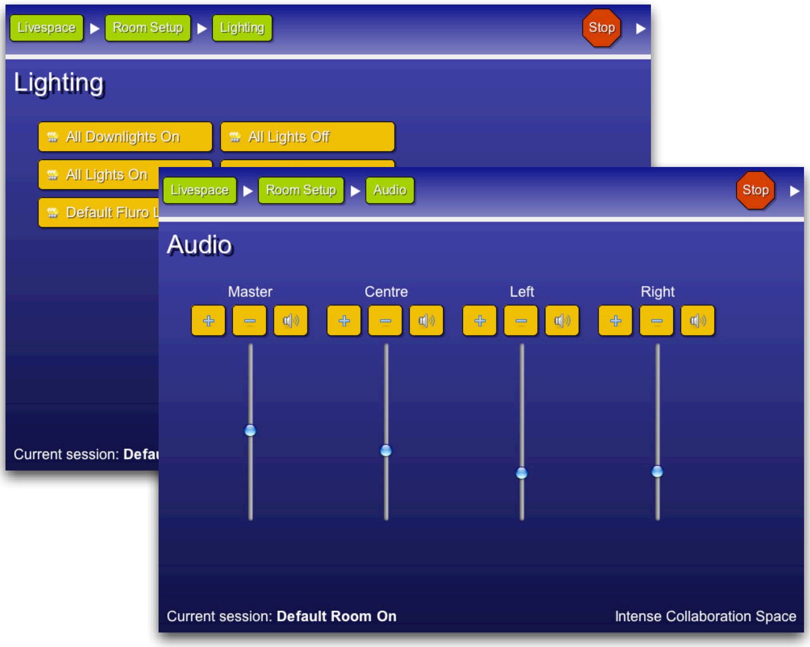
The user can switch a computer’s video to a display by simply tapping the computer and the display in any order, or restore the default video source of a display by double-tapping the display.[17] If a computer has several video outputs, as is the case when the computer’s video card has several desktops, a menu appears to allow selection of the desired desktop.

The video switching described here should not be confused with the VNC-based software screen sharing application described previously, even though they they provide some overlapping functionality. Video switching is done over video cables connected to a hardware video switch; screen sharing is done in software over the network – each approach has strengths and weaknesses. In future we plan to explore the possibility of uniting these two technologies, which would require a re-think of the user interface, since software forwarding provides capabilities not possible with hardware switching, such as being able to forward any computer to any computer.

[17] The default video sources are customisable per session.

### Other Pages

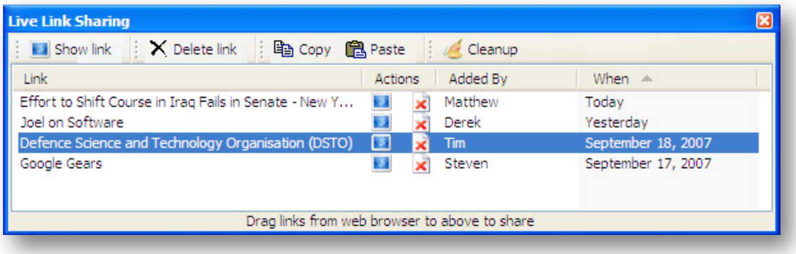
Ignite also allows the user to select lighting scenes and control the audio mixer volume levels. There is also an optional page for controlling the position of devices with attitude control motors, such as the displays used in the HxI Braccetto CTW devices (HxI 2007) or video cameras.



*The Lighting and Audio pages*

## 7.5. Link Sharing

The Link Sharing applet provides a way for people to quickly open web pages or other documents on several computers. The usual mode of use is to navigate to the page of interest in a web browser[18] and then drag the link from the browser's location field onto the link sharing window, which adds it to the list of shared links that can be seen in the screenshot below.



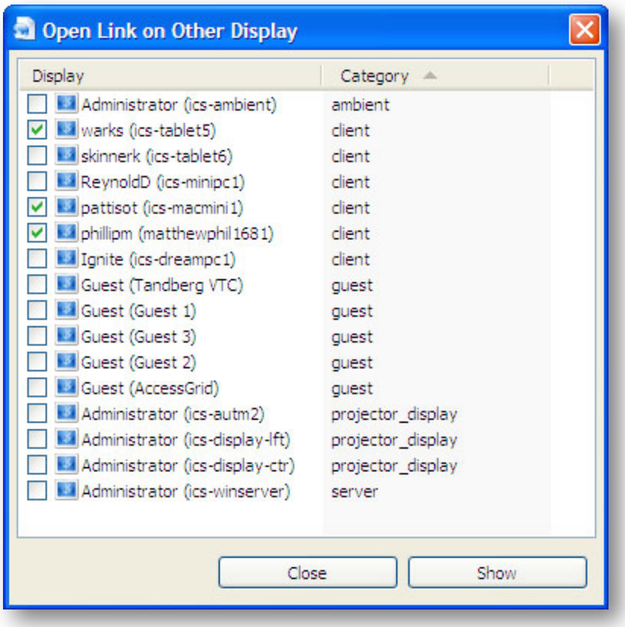
| Link | Actions | Added By | When |
|---|---|---|---|
| Effort to Shift Course in Iraq Fails in Senate - New Y... | | Matthew | Today |
| Joel on Software | | Derek | Yesterday |
| Defence Science and Technology Organisation (DSTO) | | Tim | September 18, 2007 |
| Google Gears | | Steven | September 17, 2007 |

Drag links from web browser to above to share

*The link sharing window*

[18] Links may be to a standard web page, office document, or a page in the Livespace information repository.

Once shared, a link can be opened on any number of computers by selecting it and choosing 'Show Link', which displays the window below.



| Display | Category |
|---|---|
| □ Administrator (ics-ambient) | ambient |
| ☑ warks (ics-tablet5) | client |
| □ skinnerk (ics-tablet6) | client |
| □ ReynoldD (ics-minipc1) | client |
| ☑ pattisot (ics-macmini1) | client |
| ☑ phillipm (matthewphil1681) | client |
| □ Ignite (ics-dreampc1) | client |
| □ Guest (Tandberg VTC) | guest |
| □ Guest (Guest 1) | guest |
| □ Guest (Guest 3) | guest |
| □ Guest (Guest 2) | guest |
| □ Guest (AccessGrid) | guest |
| □ Administrator (ics-autm2) | projector_display |
| □ Administrator (ics-display-lft) | projector_display |
| □ Administrator (ics-display-ctr) | projector_display |
| □ Administrator (ics-winserver) | server |

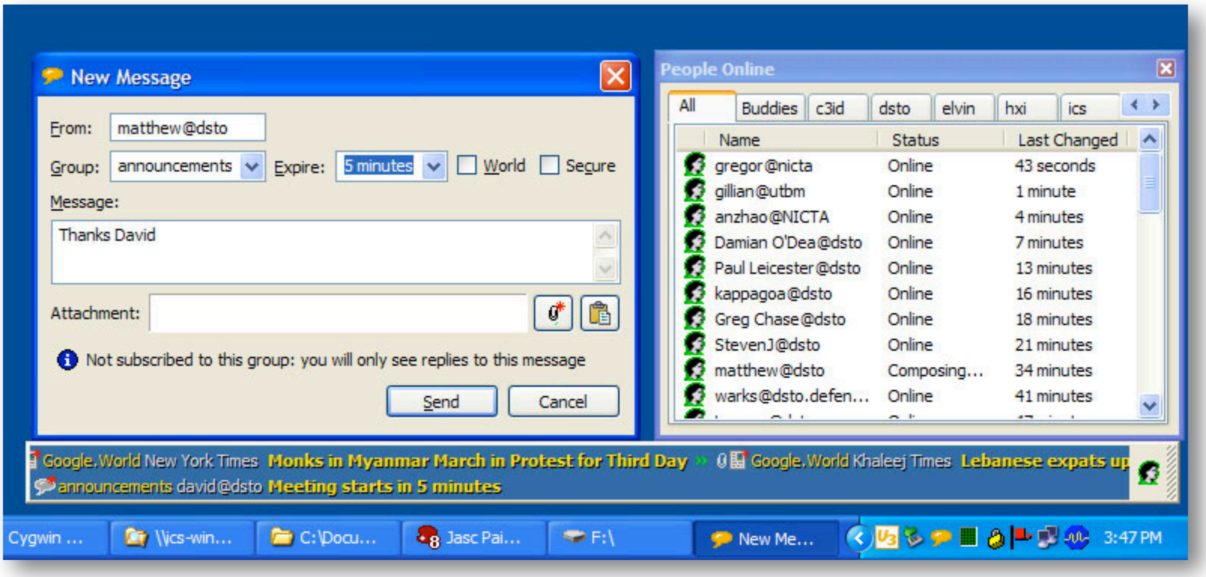
*The link sharing 'Show Link' window*

In the example, the link will be opened on the three selected computers when the 'Show' button is hit. The selected set of computers is preserved for the next time the window is opened, assisting the common use case of opening several links on the same set of computers.

The shared links persist until they're deleted, or the 'Cleanup' button can used to manually delete links older than a week, keeping at most twenty of the newest ones.

## 7.6. Sticker

Sticker provides an instant messaging and virtual presence service in a Livespace (Sticker 2007). It appears as a tickertape-style strip at the bottom of the screen: incoming messages slide across on the tape similar to those on a stock ticker.



From: matthew@dsto

Group: announcements Expire: 5 minutes □ World □ Secure

Message:

Thanks David

Attachment:

Not subscribed to this group: you will only see replies to this message

| Name | Status | Last Changed |
|---|---|---|
| gregor@nicta | Online | 43 seconds |
| gillian@utbm | Online | 1 minute |
| anzhao@NICTA | Online | 4 minutes |
| Damian O'Dea@dsto | Online | 7 minutes |
| Paul Leicester@dsto | Online | 13 minutes |
| kappagoa@dsto | Online | 16 minutes |
| Greg Chase@dsto | Online | 18 minutes |
| StevenJ@dsto | Online | 21 minutes |
| matthew@dsto | Composing... | 34 minutes |
| warks@dsto.defen... | Online | 41 minutes |

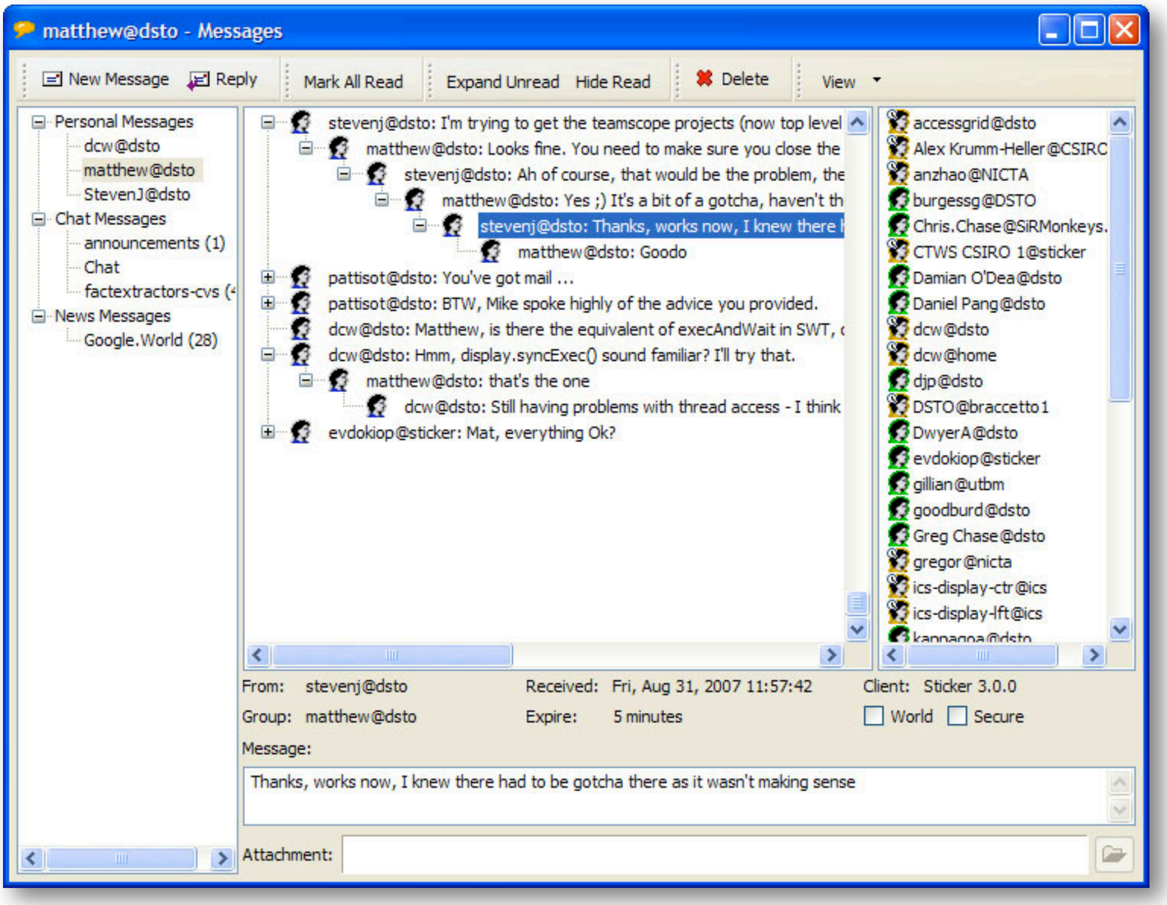
Google.World New York Times **Monks in Myanmar March in Protest for Third Day** » 0 Google.World Khaleej Times **Lebanese expats up**

announcements david@dsto **Meeting starts in 5 minutes**

*Sticker on the desktop*

People in a Livespace can send instant messages to anyone else or to pre-defined groups to which several people may belong. Messages can also be generated by automated services, such as news originating from RSS feeds or alerts notifying of events including document changes, meeting reminders, etc.

As well as the ticker interface, which uses minimal desktop space and is well suited to short news messages and announcements, Sticker also has a more traditional threaded message list view which is better suited for ongoing conversations between people.



stevenj@dsto: I'm trying to get the teamscope projects (now top level
matthew@dsto: Looks fine. You need to make sure you close the
stevenj@dsto: Ah of course, that would be the problem, the
matthew@dsto: Yes ;) It's a bit of a gotcha, haven't th
stevenj@dsto: Thanks, works now, I knew there h
matthew@dsto: Goodo
pattisot@dsto: You've got mail ...
pattisot@dsto: BTW, Mike spoke highly of the advice you provided.
dcw@dsto: Matthew, is there the equivalent of execAndWait in SWT, c
dcw@dsto: Hmm, display.syncExec() sound familiar? I'll try that.
matthew@dsto: that's the one
dcw@dsto: Still having problems with thread access - I think
evdokiop@sticker: Mat, everything Ok?

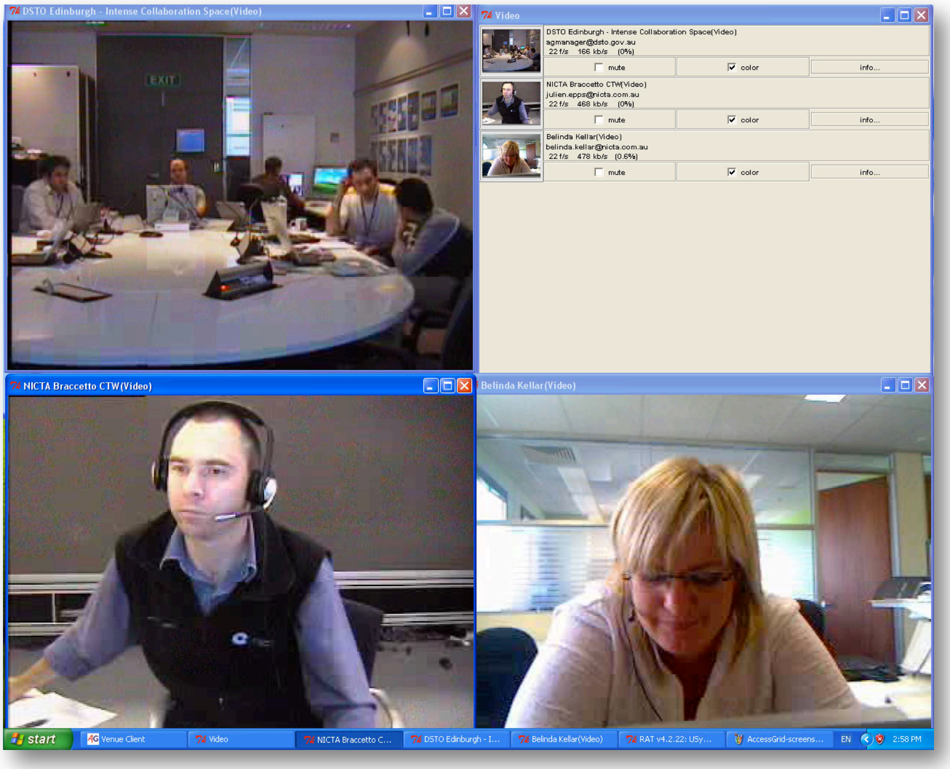
accessgrid@dsto
Alex Krumm-Heller@CSIRC
anzhao@NICTA
burgessg@DSTO
Chris.Chase@SiRMonkeys.
CTWS CSIRO 1@sticker
Damian O'Dea@dsto
Daniel Pang@dsto
dcw@dsto
dcw@home
djp@dsto
DSTO@braccetto1
DwyerA@dsto
evdokiop@sticker
gillian@utbm
goodburd@dsto
Greg Chase@dsto
gregor@nicta
ics-display-ctr@ics
ics-display-lft@ics
kappagoa@dsto

From: stevenj@dsto
Received: Fri, Aug 31, 2007 11:57:42
Client: Sticker 3.0.0
Group: matthew@dsto
Expire: 5 minutes
□ World □ Secure
Message:
Thanks, works now, I knew there had to be gotcha there as it wasn't making sense
Attachment:

*The Sticker Messages window*

## 7.7. AccessGrid

AccessGrid (AccessGrid 2007) is not a product of the Livespace project, but it has become an indispensable part of the Livespace suite. AccessGrid provides scalable software video teleconferencing capability. It is open source and available for a wide array of off-the-shelf computing platforms, making it ideal for inter-organisational collaboration where proprietary products may not be affordable, or where existing solutions don't interoperate.



DSTO Edinburgh - Intense Collaboration Space(Video)
agmanager@dsto.gov.au
22 f/s 166 kb/s (0%)
☐ mute ☑ color info...

NICTA Braccetto CTW(Video)
julien.epps@nicta.com.au
22 f/s 468 kb/s (0%)
☐ mute ☑ color info...

Belinda Kellar(Video)
belinda.kellar@nicta.com.au
22 f/s 478 kb/s (0.6%)
☐ mute ☑ color info...

*An AccessGrid session*

AccessGrid provides a set of virtual 'Venues'. Entering a venue, assuming you have been granted access, allows you to engage the other members of the venue in a video teleconference. It also supports simple text messaging and some shared applications, such as a shared presentation viewer.

The minimum hardware needed to use AccessGrid is a PC with an IP network connection, a basic USB webcam (although video quality will be substantially improved by a higher-quality video camera), and either an omni-directional microphone with echo cancelling, or a microphone headset. AccessGrid transmits audio and video via multicast UDP/IP, so for networks where this is not configured (and WAN environments) a multicast-to-unicast bridge service will be needed. AccessGrid sessions are coordinated by a 'Venue Server', one of which will need to be available on the same network.

AccessGrid has been superficially integrated into Livespaces so that an AccessGrid VTC can be automatically launched as part of a Livespace session. In future we would like investigate how we can further integrate it by storing window layouts and other configuration.

# 8. Administrative Applications

In addition to the applications listed above that support the end users of a Livespace, a number of administrative tools are available which help the managers of a Livespace to effectively configure and maintain the system.

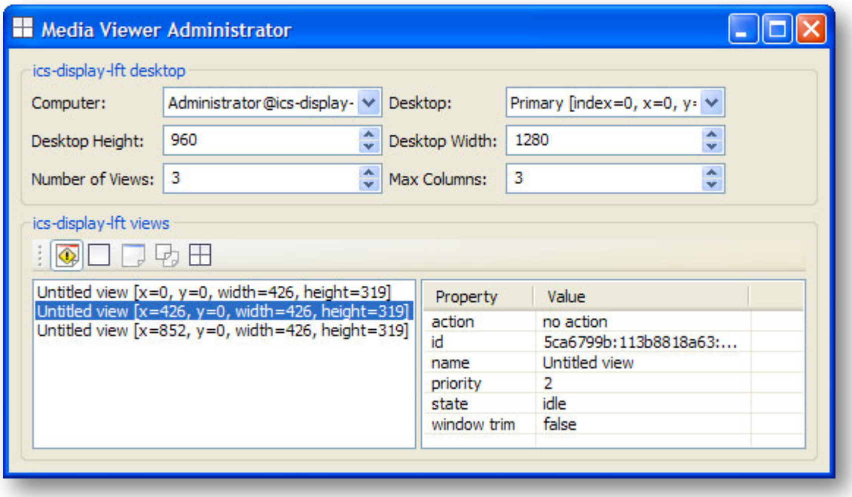
## 8.1. Service Browser

The service browser application is an indispensable debugging and exploration aid, providing a view of all entities published on the Livespace Bus. An example of its use has previously been demonstrated in the 'Livespace Bus' section.

As well as providing a way to browse the entities in a Livespace, the browser can be used to edit their state, which can be useful when exploring behaviour, or when no other interface is available.

## 8.2. Media Viewer

The media viewer application is used to set up viewer portals that contain the media service's multimedia presentations on a computer's desktop. The default is to simply have one portal encompassing the entire desktop, however the media viewer can be used to partition the space into several portals, allowing several media displays at once. For example a video display on one half of a screen and a PowerPoint presentation on the other half.



ics-display-lft desktop

Computer: Administrator@ics-display-

Desktop: Primary [index=0, x=0, y=

Desktop Height: 960

Desktop Width: 1280

Number of Views: 3

Max Columns: 3

ics-display-lft views

Untitled view [x=0, y=0, width=426, height=319]
Untitled view [x=426, y=0, width=426, height=319]
Untitled view [x=852, y=0, width=426, height=319]

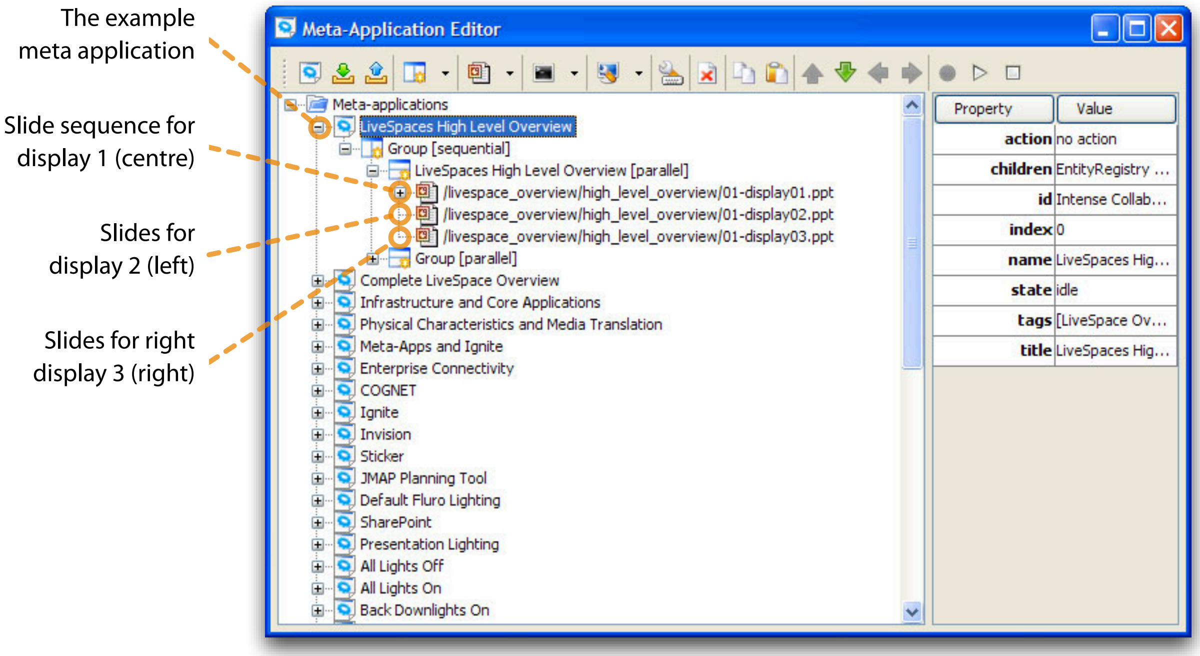
| Property | Value |
|---|---|
| action | no action |
| id | 5ca6799b:113b8818a63:... |
| name | Untitled view |
| priority | 2 |
| state | idle |
| window trim | false |

*The Media Viewer window*

As well as potentially making better use of screen real-estate, this capability is crucial when authoring meta applications on a single desktop, which are intended for a room with multiple displays – meta application developers can test their multi-display meta applications all on one screen with several portals before deploying to the room.

## 8.3. The Meta Application Editor

The Meta Application editor is the authoring tool that is used to script the series of coordinated actions that make up a meta application. It provides the ability to browse and edit meta applications, and to start, pause and stop them.



The example meta application

Slide sequence for display 1 (centre)

Slides for display 2 (left)

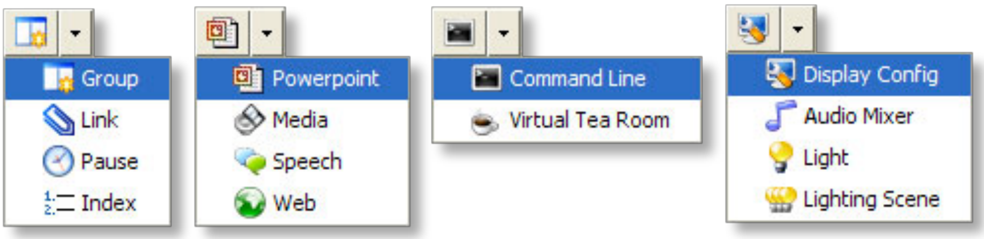
Slides for right display 3 (right)

Meta-Application Editor
Meta-applications
LiveSpaces High Level Overview
Group [sequential]
LiveSpaces High Level Overview [parallel]
/livespace_overview/high_level_overview/01-display01.ppt
/livespace_overview/high_level_overview/01-display02.ppt
/livespace_overview/high_level_overview/01-display03.ppt
Group [parallel]
Complete LiveSpace Overview
Infrastructure and Core Applications
Physical Characteristics and Media Translation
Meta-Apps and Ignite
Enterprise Connectivity
COGNET
Ignite
Invision
Sticker
JMAP Planning Tool
Default Fluro Lighting
SharePoint
Presentation Lighting
All Lights Off
All Lights On
Back Downlights On

| Property | Value |
|---|---|
| action | no action |
| children | EntityRegistry ... |
| id | Intense Collab... |
| index | 0 |
| name | LiveSpaces Hig... |
| state | idle |
| tags | [LiveSpace Ov... |
| title | LiveSpaces Hig... |

*The Meta Application Editor*

The editor's window displays the room's available meta applications in a list on the left hand pane. Meta applications can be selected and expanded to see their structure, which consists of a hierarchy of actions to be executed.



*The meta application action types*

The actions that can be executed by a meta application are shown as they appear in their menus above. The actions provide the ability to:

- *Group* a series of actions for sequential or parallel execution

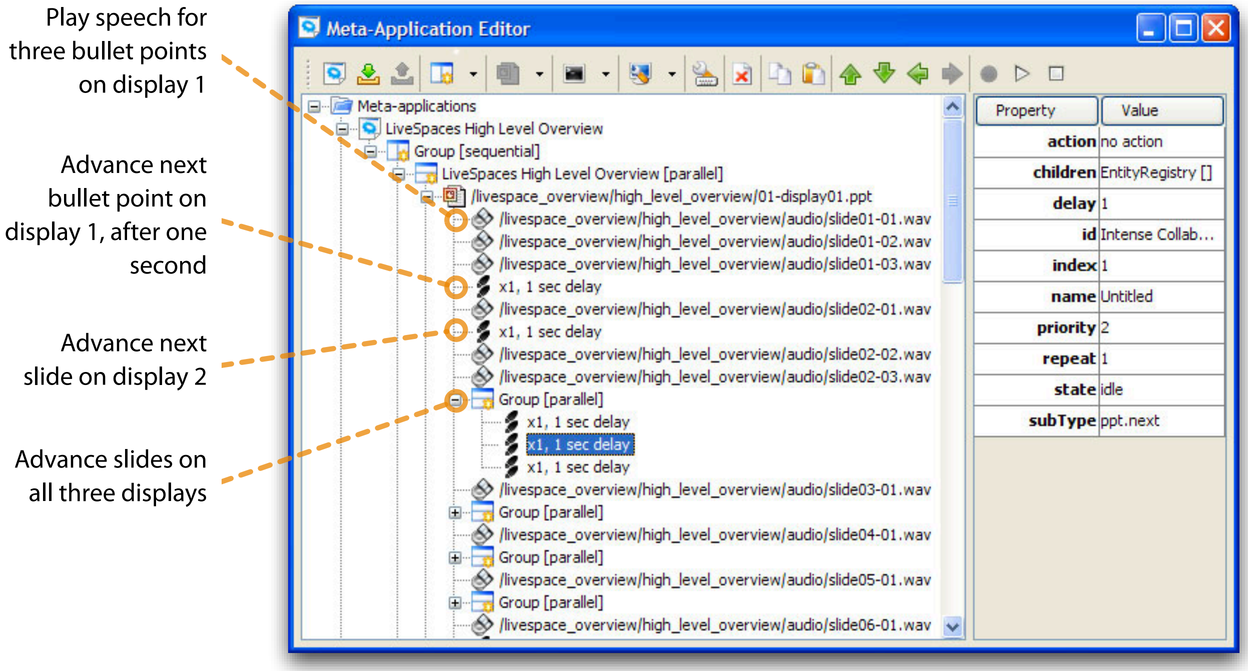
- *Link* to another meta application
- *Pause* the execution of the application for a pre-set amount of time
- Display *PowerPoint presentation slides*, with optional control of stepping through their transitions
- Play audio/visual *media* files
- Generate *speech* from text
- Open a *web page* or any other document via a URI
- Run an application on a given computer using a *command line*
- Control the room's video switching (*Display Config*), *audio mixing* or *lighting*
- Set any property on any entity, which is the fallback when none of the other specialised actions does what is needed

## An Example Meta Application

Meta applications can be used for many purposes, but one common application is for automated multimedia presentations, which are typically based on slides, with optional supporting text-to-speech audio, still pictures, and video graphics. To illustrate how this sort of meta application is scripted in the editor, we'll run through the 'Livespaces High Level Overview' meta application, which is a short presentation about Livespaces itself.

In the example meta application (selected in the screenshot above), there are two groups inside a top-level group. The two child groups execute the two main phases of the presentation one after the other: the main presentation and a conclusion. The first group has three actions that run Microsoft PowerPoint slide presentations simultaneously, each on a different display in the room. The second group, which runs when the first is complete, concludes the presentation by displaying a 'Questions?' slide and some graphics.

Expanding the main presentation group shows the structure below:



Play speech for three bullet points on display 1

Advance next bullet point on display 1, after one second

Advance next slide on display 2

Advance slides on all three displays

```
Meta-applications
  LiveSpaces High Level Overview
    Group [sequential]
      LiveSpaces High Level Overview [parallel]
        /livespace_overview/high_level_overview/01-display01.ppt
          /livespace_overview/high_level_overview/audio/slide01-01.wav
          /livespace_overview/high_level_overview/audio/slide01-02.wav
          /livespace_overview/high_level_overview/audio/slide01-03.wav
          x1, 1 sec delay
          /livespace_overview/high_level_overview/audio/slide02-01.wav
          x1, 1 sec delay
          /livespace_overview/high_level_overview/audio/slide02-02.wav
          /livespace_overview/high_level_overview/audio/slide02-03.wav
          Group [parallel]
            x1, 1 sec delay
            x1, 1 sec delay
            x1, 1 sec delay
          /livespace_overview/high_level_overview/audio/slide03-01.wav
          Group [parallel]
          /livespace_overview/high_level_overview/audio/slide04-01.wav
          Group [parallel]
          /livespace_overview/high_level_overview/audio/slide05-01.wav
          Group [parallel]
          /livespace_overview/high_level_overview/audio/slide06-01.wav
```

| Property | Value |
|---|---|
| action | no action |
| children | EntityRegistry [] |
| delay | 1 |
| id | Intense Collab... |
| index | 1 |
| name | Untitled |
| priority | 2 |
| repeat | 1 |
| state | idle |
| subType | ppt.next |

*The internal structure of a meta application*

The first screenshot showed how three slide presentations were set up to drive the three displays.[19] In the second screenshot you can see how the rest of meta application is driven: by a series of actions under the main slide-show action.

After showing the first slide of the three slide packs on the three displays, the meta application proceeds as follows:

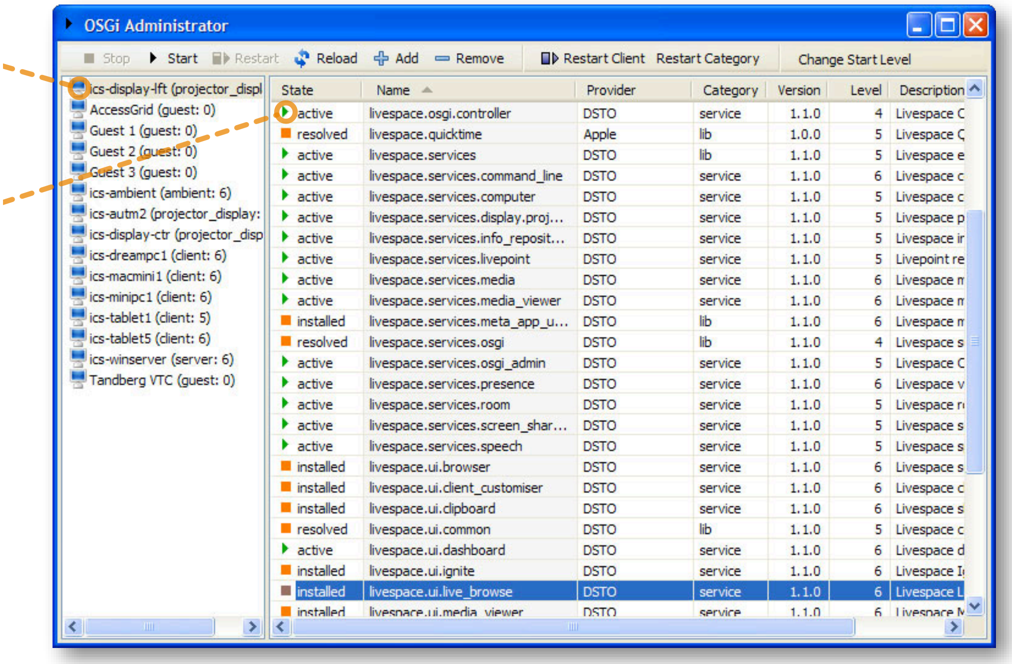
- The display 1 introduction slide is accompanied by a voice-over generated by a text-to-speech (TTS) engine. In this case the TTS is pre-generated and stored as audio files, but it could also be generated on-the-fly.
- After the initial bullet point is shown and the introduction read, the main presentation is advanced one step to show the next bullet point, followed by a voice-over for the new text.
- All three slide shows are then advanced to the next slide using a parallel group with three 'step' actions. Note in the screenshot how the step actions are targeted at a display using the 'priority' field – although the step actions are children of the display 1 presentation, they can target the other displays. This is a useful way to coordinate actions on separate displays.

[19] Meta application displays are referred to by number (1, 2, 3, ...), which reflect their relative priority in the room. The centre display is usually display 1, left is 2, right is 3, etc. If a room does not have a display mapped for a priority (for example, if it only has two main displays), the actions for that priority are not executed. Equally, several displays could be mapped to the same number in order to show the same information at the same time.

The meta application then continues until stopped or it completes.

## 8.4. OSGi Administration

The OSGi Administration application controls the OSGi service bundles running on any computer in the Livespace. The computers running the Livespace OSGi client software are shown in a list on the left – selecting a computer shows its installed service bundles and their current state.



Computers in the Livespace

OSGi services running on the selected computer

ics-display-lft (projector_displ
AccessGrid (guest: 0)
Guest 1 (guest: 0)
Guest 2 (guest: 0)
Guest 3 (guest: 0)
ics-ambient (ambient: 6)
ics-autm2 (projector_display:
ics-display-ctr (projector_disp
ics-dreampc1 (client: 6)
ics-macmini1 (client: 6)
ics-minipc1 (client: 6)
ics-tablet1 (client: 5)
ics-tablet5 (client: 6)
ics-winserver (server: 6)
Tandberg VTC (guest: 0)

| State | Name | Provider | Category | Version | Level | Description |
|---|---|---|---|---|---|---|
| active | livespace.osgi.controller | DSTO | service | 1.1.0 | 4 | Livespace C |
| resolved | livespace.quicktime | Apple | lib | 1.0.0 | 5 | Livespace Q |
| active | livespace.services | DSTO | lib | 1.1.0 | 5 | Livespace e |
| active | livespace.services.command_line | DSTO | service | 1.1.0 | 6 | Livespace c |
| active | livespace.services.computer | DSTO | service | 1.1.0 | 5 | Livespace c |
| active | livespace.services.display.proj… | DSTO | service | 1.1.0 | 5 | Livespace p |
| active | livespace.services.info_reposit… | DSTO | service | 1.1.0 | 5 | Livespace ir |
| active | livespace.services.livepoint | DSTO | service | 1.1.0 | 5 | Livepoint re |
| active | livespace.services.media | DSTO | service | 1.1.0 | 6 | Livespace m |
| active | livespace.services.media_viewer | DSTO | service | 1.1.0 | 6 | Livespace m |
| installed | livespace.services.meta_app_u… | DSTO | lib | 1.1.0 | 6 | Livespace m |
| resolved | livespace.services.osgi | DSTO | lib | 1.1.0 | 4 | Livespace s |
| active | livespace.services.osgi_admin | DSTO | service | 1.1.0 | 5 | Livespace C |
| active | livespace.services.presence | DSTO | service | 1.1.0 | 6 | Livespace v |
| active | livespace.services.room | DSTO | service | 1.1.0 | 5 | Livespace r |
| active | livespace.services.screen_shar… | DSTO | service | 1.1.0 | 5 | Livespace s |
| active | livespace.services.speech | DSTO | service | 1.1.0 | 5 | Livespace s |
| installed | livespace.ui.browser | DSTO | service | 1.1.0 | 6 | Livespace s |
| installed | livespace.ui.client_customiser | DSTO | service | 1.1.0 | 6 | Livespace c |
| installed | livespace.ui.clipboard | DSTO | service | 1.1.0 | 6 | Livespace s |
| resolved | livespace.ui.common | DSTO | lib | 1.1.0 | 5 | Livespace c |
| active | livespace.ui.dashboard | DSTO | service | 1.1.0 | 6 | Livespace d |
| installed | livespace.ui.ignite | DSTO | service | 1.1.0 | 6 | Livespace I |
| installed | livespace.ui.live_browse | DSTO | service | 1.1.0 | 6 | Livespace L |
| installed | livespace.ui.media_viewer | DSTO | service | 1.1.0 | 6 | Livespace M |

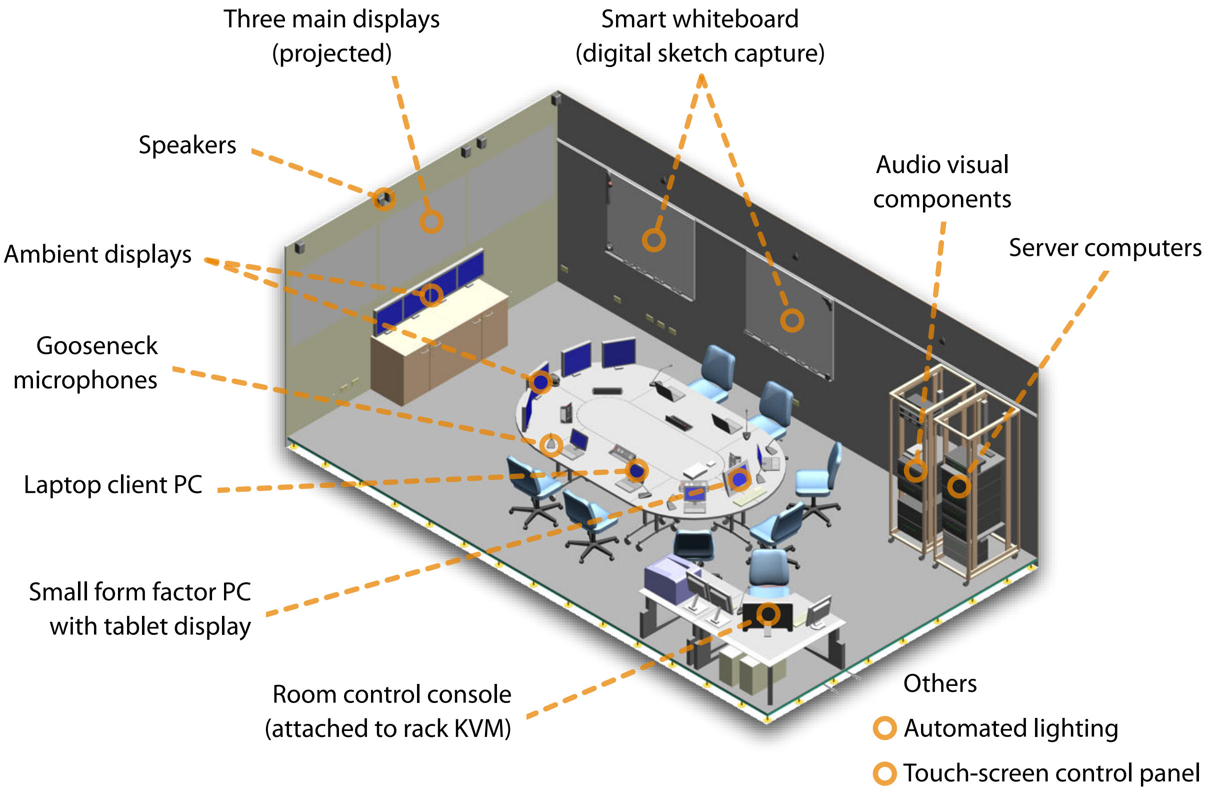
*The OSGi service administration window*

From this window, services can be started and stopped, reloaded (which forces the bundle to be re-loaded from the server), or added and removed. The administration tool can also instruct clients to restart themselves, restarting an individual client, a categories of client (servers, displays, etc.), or simply all clients in the room.

Coupled with the centralised deployment and configuration system described earlier, this remote administration capability has proven very effective for managing several rooms containing many devices.

# 9. Livespace Hardware Components

This section describes in some detail the sorts of devices that we have employed in a Livespace.


Three main displays (projected)
Smart whiteboard (digital sketch capture)
Speakers
Audio visual components
Server computers
Ambient displays
Gooseneck microphones
Laptop client PC
Small form factor PC with tablet display
Room control console (attached to rack KVM)
Others
Automated lighting
Touch-screen control panel

*Components of the Intense Collaboration Space*

Above is an isometric cutaway view of the Intense Collaboration Space (ICS) at DSTO. The ICS is our main fixed experimental facility and contains examples of nearly all the off-the-shelf technologies we've identified as potentially useful in a Livespace. The key components in the ICS are discussed below.

**Client computers**. We have trialled two sorts of computer for personal computing on the meeting table: laptops and small form-factor PCs. The key advantage of a laptop is portability, giving the ability to pick it up and move it around the room: however the fact that the laptop needs to be plugged into heavy cables means that this portability is mostly lost in the ICS – the combination of power connection, video cable (if we want to send its video to the main displays), and network cable (when being used in an environment where WiFi is not an option) makes for a formidable physical and psychological barrier to mobility.

Small form factor PCs, which are much cheaper than laptops, and generally faster and more reliable, have turned out to be a better choice for a permanent facility such as the ICS. When coupled with a digitising tablet (such as those sold by Wacom) and a suitable operating system such Windows XP Tablet Edition or Mac OS X, these PCs can also accept pen input if required. PC units can either be deployed on the table as in the ICS, or stored in a rack, with their keyboard, mouse and video extended out to the table via a KVM extender solution.

**Server computers**. A room like the ICS will only need a single server computer, which provides the core software services (bundle deployment, event routing, HTTP/WebDAV server and shared file system) needed for a Livespace. The server may also act as a driver for peripheral devices, such as the Clipsal C-Bus automation system used for lighting control in the ICS, or to manage one or more serial port interfaces used to control projectors, etc.

**Shared displays**. Large scale, shared displays provide an important space for displaying information and collaborating in a Livespace. Depending on the size of the room, shared displays can be generated by projectors or large plasma/LCD screens. In the ICS we use three ceiling-mounted consumer-grade projectors throwing their displays onto the wall in the front of the room.

Optionally, one or more 'ambient' displays might be deployed in the room: these displays are typically smaller and less centrally located than the shared displays. Ambient displays are often used for showing background information, such as status displays, tickertape news, and virtual presence rosters.

**Display computers**. If a Livespace has shared displays, it can also be useful to have one or more computers dedicated to driving those displays – the alternative is to be limited to switching the video from the client computers to the main displays. Two common configurations are to have one computer per display (a separate desktop per display), or one computer with several desktops driving all the displays (one desktop stretched over several displays).

In the ICS we have three display computers, one per projector, and typically use the one-desktop-per-display arrangement. However, since the centre display server has a triple-head video card, it can be reconfigured into the single-desktop arrangement as needed.

The single desktop model has the advantage of allowing windows to be dragged across displays. The main disadvantage is many applications are badly-behaved in the presence of multiple displays, showing pop-up windows on the wrong display, miscalculating the display size, and other problems.

**Audio/Video switch**. The audio and video outputs of the computers in the room can be connected to a switch in the room's equipment rack, and automatically controlled by Livespace driver services. This provides the ability to switch video from the client and display computers to the shared displays, as well as switch, mix, and set the volume of audio in the room.

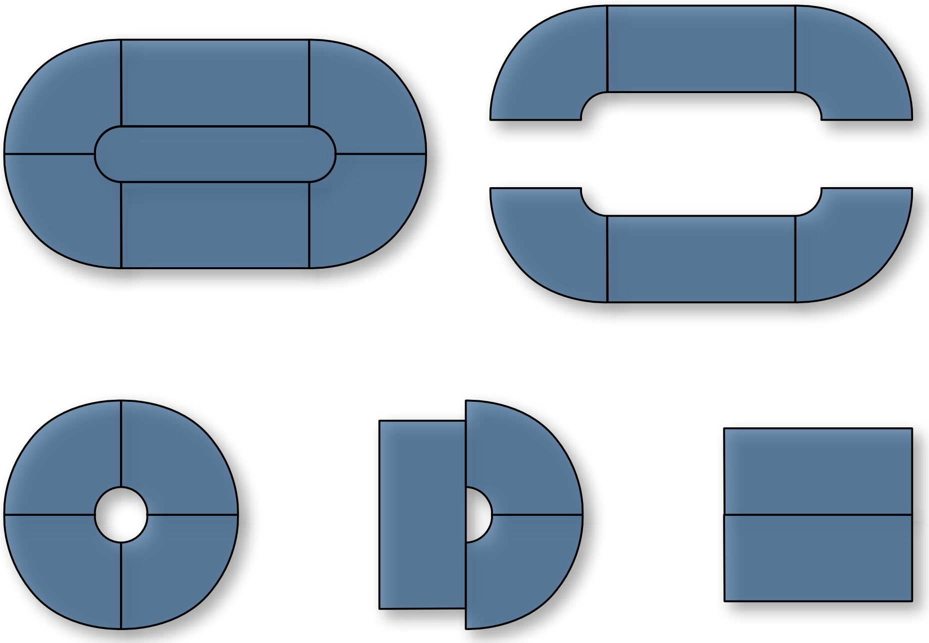
**Keyboard/Video/Mouse switch**. In the ICS we have a dedicated LCD display and keyboard console connected to a KVM, which can be used to control the server and display computers in the room. The KVM console is the natural place for the room's administrator to operate from.

**Room control panel**. Ideally an ICS-like room will have a panel at the entrance running Ignite to make it easy to set up the room. This panel can be a simple LCD touch screen mounted just inside the entrance, connected to a computer in the rack which is running Ignite in full-screen 'kiosk' mode. The computer driving the display can either be a dedicated, low spec PC, or the room's main server, since its display is generally not used for anything else when not being employed by the room's administrator. The touch screen we use in the ICS connects to the computer via USB and generates mouse inputs in response to finger drags and taps.

**Lights**. The ICS lighting is provided by down-lights and conventional fluorescent strip lighting, both of which can be automatically controlled via a C-Bus automation unit. The key benefit of automated lighting is it allows Livespace sessions and meta applications to switch room lighting scenes, for example to switch lights on when the room starts, change to presentation lighting for automated briefs, and then change back to an appropriate scheme for a meeting.

**Smart whiteboards**. The ICS includes fours whiteboards mounted on sliding brackets, to which smart whiteboard digitisers have been attached to capture sketches.[20] When someone starts drawing on a board, an associated application launches on the display on the same side of the room as the board and shows the captured diagram. The result can be saved or printed from the application. Some models can also provide optical character recognition of any hand-written text.

**Modular Table**. The main meeting table in the ICS is composed of seven re-configurable modules, which can be arranged into several shapes as needed.[21] The tables are also finished with a smooth lacquer that can be drawn on with conventional whiteboard markers, so people can simply scribble on the table if that suits them – typically these sorts of outputs are captured with a digital camera after the meeting.



*Some of the arrangements possible with the ICS modular desks*

**Video conferencing.** We have experimented with several types of video conferencing technology in the ICS: AccessGrid and ConferenceXP (ConferenceXP 2007), both of which are software

[20] To date, we have trialled two types of whiteboard capture device, eBeam (eBeam 2007) and Mimio (Mimio 2007).

[21] In theory anyway: in practice, the sheer number of cables and wires connected to the devices on the table, and between the table modules, mean that re-configuring the table is a rare event.

packages that run on off-the-shelf PC hardware (typically a PC plus USB camera and echo cancelling microphone or headset) and communicate over IP, and a Tandberg MXP6000 hardware unit, which provides a turnkey solution that can use IP or ISDN.[22]

While hardware solutions offer advantages in ease of setup and use, and often provide better video quality, they are also an order of magnitude more expensive to deploy than software solutions, and are far less customisable – for this reason, in an experimental space such as the ICS, we have found the software solutions to be more appropriate. The free software solutions are also easier to deploy when collaborating with third parties who may not have the budget for a hardware unit.

**Audio capture**. Various microphones (headset, gooseneck, and table-mounted omni-directional) are available in the room and, coupled with an audio switch/mixer, echo canceller and audio digitiser, these can be used for various purposes including VTC, audio recording and speech-to-text transcription.

**Other infrastructure**. Wires and cables are the bane of a facility with a large number of electronic devices.[23] In the ICS we have tried to mitigate the problem by running the majority of wires under the raised floor – even so, the tables become effectively bolted to the floor once all the cables are connected, so strategies for reducing the number of cables going to the table are high on the requirements list for future iterations.

One way to reduce the cabling that has already been employed in the newer Battlelabs is to move the client computers off the table into a rack, and use a Keyboard/Video/Mouse (KVM) extender solution to connect the screen, keyboard and mouse on the table via a dedicated CAT 5 twisted pair cable or via the network over IP.[24] This eliminates the two thickest cables (power and video), replacing them with single, thin CAT 5 wire terminated in a KVM box on the table.

[22] We have also experimented with other consumer-oriented products such as Skype but, as discussed in the introduction to this paper, most of these do not work well (or at all) in multi-room/multi-participant scenarios.

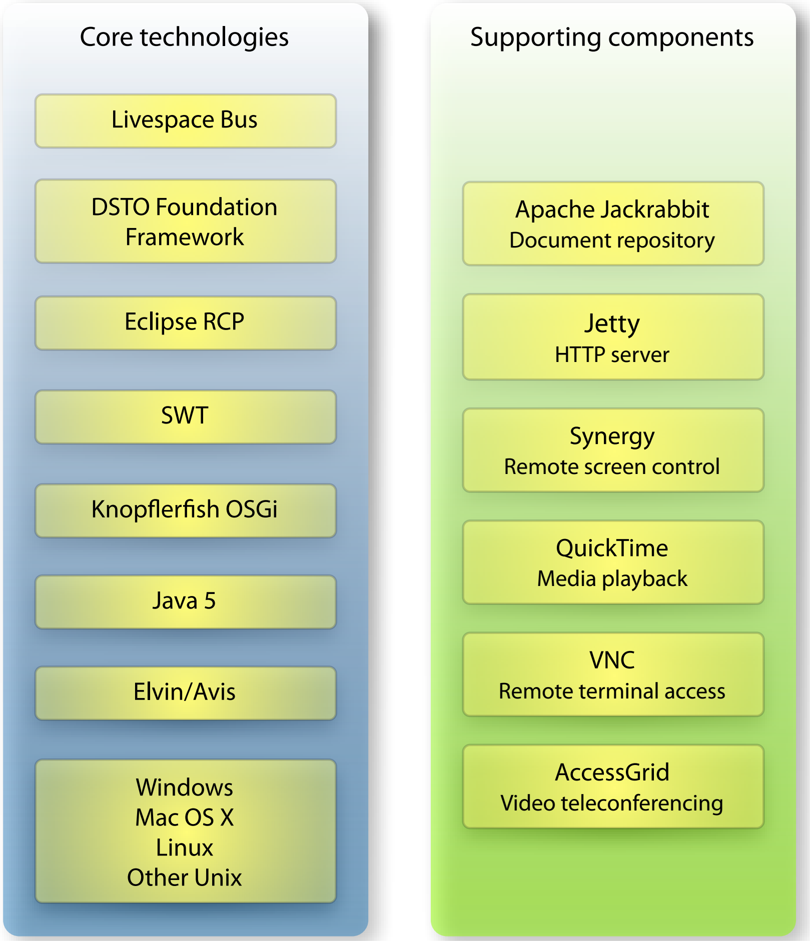
[23] This situation is exacerbated in the Defence classified domain where wireless communications are usually not permitted.

[24] In a classified environment this arrangement also has the advantage that all computing devices can be housed in a single secure rack, or even located in a secure facility away from the Livespace.

# 10. Foundation Technologies

The Livespaces environment has been engineered from existing DSTO software components and a number of components from third parties. The platform is Java-based, with the Eclipse Rich Client Platform (Eclipse 2007) providing the framework for the desktop applications. Wherever possible we have selected open source components to reduce licensing issues on deployment and to allow us to customise the software to our needs. This choice of technologies enables us to deploy Livespaces in a wide variety of environments, which widens the scope for experimentation and helps us to employ best-of-breed technology wherever we can find it.

The diagram below shows the core Livespace software component stack, and the key supporting 3rd party components that we build on to provide essential services.



*Technology foundations for Livespaces*

# 11. History

Livespaces arose out of DSTO research into supporting intense collaboration during the planning phase of a military operation. In this sort of collaboration, expert planners are called in to define and evaluate the best courses of action to achieve their commander's stated intent for an operation. The planning phase of an operation usually has a fixed time limit, especially in the disaster rescue and peacekeeping missions often required of the Australian Defence Force (ADF). It requires the staff to collaborate at a high level for a sustained period in order to make most effective use of their limited time and (often) incomplete information.

The Livespaces program draws from previous work in the Computer Supported Cooperative Work research area in defining the concept of Intelligent Interactive Workspaces (Vernik *et al.* 2003). This research looked into methods of supporting collaboration between multiple participants in an enterprise, rather than just supporting the work practices of individuals.

Prototyping initiated with the Augmented Synchronised Planning Spaces (Ausplans) project, which was a joint effort between DSTO, the University of South Australia (UniSA) and the Distributed Systems Technology Centre (DSTC) (R. Vernik *et al.* 2004; Evdokiou *et al.* 2004). The Ausplans project began with a prototype laboratory built by combining technologies from MIT (Meta Glue), Stanford (iROS), and the DSTC (Elvin and ODSI): the resulting prototype was the first 'LiveSpace', although the underlying technology was very different to what came later.

Several foundational experiments were conducted using the prototype smart meeting space at UniSA with the advice and participation of experienced planning staff from Deployable Joint Force Headquarters (DJFHQ), based in Brisbane. Based on the outcome of these experiments and our experiences gained in integrating the prototype technology at UniSA, work began on developing a fully-integrated system to support smart meeting rooms that would meet the requirements of ADF planning staff – this development marked the start of the Livespaces technology as it is described in this document.

Although the Livespaces research and development has been driven by the needs of the ADF, the kinds of collaboration it supports clearly have general application. The cross-organisation research being conducted as part of the HxI Braccetto project aims to identify how best to apply the research results and technologies across domains.

## 11.1. Livespaces Timeline

2002 Initial research at University of South Australia (Vernik, Bright, Blackburn) with experiments using MIT Meta Glue for future smart teamspaces. Based on iROS (Stanford) & ODSI/Elvin (DSTC)

2003 First Livespace environment established at UniSA. Relationship established with Stanford University and the use of iROS as the operating environment with workspaces and ODSI/Elvin for enterprise integration. Augmented Synchronised Planning Spaces (Ausplans) project started.

2004 DSTC SA established. ICS built at DSTO Edinburgh. Livespaces used as part of the Augmented Synchronised Planning Spaces project. First experimentation using actual planning teams. Research conducted into workspace session interfaces and workspace simulation. Session interface research leads to first Ignite prototype. Development of new Livespace operating environment started.

2005 DSTC Ausplans project complete. DSTO FOCAL facility enhanced with Livespaces capabilities. New Livespace labs established at Deployable Joint Force Headquarters (DJFHQ) in Brisbane. Final experimentation undertaken as part of Exercise Pozieres Development '05 conducted at DJFHQ in November.

2006 DSTO Livespace operating environment in place, remaining iROS and ODSI components retired. Livespaces integrated as part of the HxI Braccetto project. Command TeamNets established based on Livespaces technologies and concepts.

2007 Coalition TeamNets established to support coalition experimentation. New Composable Collaboration Systems laboratory established. Livespaces 1.0 and 1.1 baselined and deployed to newly-established Australian Battlelabs and Canadian DRDC.

# 12. Current Status And Future Work

As of November 2007, Livespaces is about to reach its second major baseline, release 1.1. While the 1.0 baseline was targeted at quality and ease of deployment, release 1.1 represents a series of incremental improvements across all components, as well as two major new components: a revamped Ignite interface and the Information Repository.

The Livespaces 1.0 baseline has been deployed to several Defence sites around Australia, and three installations established by DRDC in Canada. It has also become part of the framework supporting the HxI Braccetto collaborative telepresence workstations (CTWs), operated by DSTO, CSIRO and NICTA.

While we have many items of work scheduled for an anticipated 1.2 baseline, the Livespaces infrastructure has reached a fairly mature and stable state, and future work on Livespaces will now be dependent on outcomes of our research into user interfaces for distributed awareness and collaboration. In terms of purely extending Livespaces capabilities, we would like to explore more sophisticated interfaces for controlling Livespaces, aimed at providing a holistic visualisation of rooms, people and services, and allowing them to be manipulated directly within the interface. We have already begun exploring, via a series of storyboards, advanced interfaces for visualising and managing multiple-room screen forwarding.

We also intend to make the Livespaces infrastructure more widely available, most likely under an open source licensing arrangement. While we currently rely on unilateral agreements between our research partners for collaboration, we would like to stimulate wider interest and research in this area – we think making the Livespace framework an open standard is the best way to begin achieving that. Developing Livespaces to its current level would not have been possible without the support of a number of open source components, and this would also be a good way of making a contribution in kind.

## Key Contributors

A number of people have contributed to the research and development that has resulted in Livespaces.

Dr. Rudi Vernik. Research leader.

Matthew Phillips. Lead developer. Designer of the Livespace Bus and core services.

Derek Weber. Key developer. Responsibilities include screen sharing and lighting control.

Steven Johnson. Key developer. Responsibilities include the Session Manager and Information Repository.

Peter Evdokiou. Researcher.

David Karunaratne. Livespaces Developer. Responsibilities include the Meta Application system and link sharing.

## Acknowledgements

The style and content of this paper has greatly benefited from the review and advice of Derek Weber, David Karunaratne and Tim Pattison.

# Abbreviations

| | |
|---|---|
| ADF | Australian Defence Force. |
| CAT 5 | Category 5 shielded twisted pair cable. |
| CORBA | Common Object Request Broker Architecture, the OMG's distributed object communication system. |
| CRUD | Create Retrieve Update Delete, the four fundamental operations performed by database applications. |
| CSIRO | Commonwealth Science and Industrial Research Organisation. |
| CTW | Collaborative Telepresence Workstation, built as part of the HxI initiative. |
| DCOM | Microsoft's Distributed Component Object Model (also known as COM+). |
| DJFHQ | ADF Deployable Joint Forces Headquarters, located in Brisbane, Australia. Now known as HQ1DIV. |
| DRDC | Defence Research Establishment Valcartier, Canada. |
| DSTC | Distributed Systems Technology Centre, a cooperative research centre, headquartered in Brisbane, which operated from 1992 to 2006. |
| DSTO | Australian Defence Science and Technology Organisation, headquartered in Edinburgh, South Australia. |
| FOCAL | Future Operations Centre Analysis Laboratory, an experimental facility located at DSTO Edinburgh. |
| GUID | Globally-unique identifier. |
| HCI | Human-computer interaction. |
| HTTP | Hypertext Transfer Protocol. |
| HxI | A research initiative involving collaboration between DSTO, CSIRO and NICTA. The HxI Initiative develops technologies that support humans in their interactions with each other, their environments and information. |
| ICS | The Intense Collaboration Laboratory, located at DSTO Edinburgh. |
| IP | Internet Protocol. |
| iROS | The Interactive Room Operating System, a project initiated at Stanford University, USA. |
| ISDN | Integrated Services Digital Network. |
| KVM | Keyboard/Video/Mouse switch. |
| LAN | Local Area Network. |

| | |
|---|---|
| LCD | Liquid Crystal Display. |
| NICTA | National Information and Communications Technologies Australia. |
| ODSI | Open Distributed Services Infrastructure, a distributed services framework developed by DSTC using Elvin. |
| OMG | Object Management Group. |
| OSGi | The OSGi Alliance's open standard for component-based software services. |
| PC | Personal Computer. |
| RCP | The Eclipse Rich Client Platform, a user interface framework for developing desktop applications in Java. |
| REST | Representation State Transfer. |
| RPC | Remote Procedure Call. |
| RSS | Really Simple Syndication. |
| SCM | Software Configuration Management. |
| SDO | Service Data Objects. |
| SOAP | Simple Object Access Protocol. |
| SWT | Standard Widget Toolkit, a component of the Eclipse Rich Client Platform. |
| TTS | Text to speech. |
| UDP/IP | User Datagram Protocol over Internet Protocol. |
| UniSA | The University of South Australia. |
| URI | Universal Resource Indicator, a superset of Uniform Resource Locator (URL) and Uniform Resource Name (URN). |
| USB | Universal Serial Bus. |
| VNC | Virtual Network Computing. |
| VoIP | Voice over IP. |
| VTC | Video Teleconferencing. |
| WAN | Wide Area Network. |
| WebDAV | Web-based Distributed Authoring and Versioning. |
| XML | Extensible Mark-up Language. |

# References

AccessGrid (2007) *AccessGrid.org*, viewed 8 October 2007, http://www.accessgrid.org.

Apache (2007) *Apache Jackrabbit*, viewed 8 October 2007, http://jackrabbit.apache.org/.

Beatty, J., Brodsky, S., Nally, M., Patel, R. (2003) 'Next-Generation Data Programming: Service Data Objects', whitepaper, available at http://www.bea.com/dev2dev/assets/sdo/Next-Gen-Data-Programming-Whitepaper.pdf.

C-Bus (2007) *Clipsal Integrated Systems*, viewed 8 October 2007, http://www3.clipsal.com/cis/portal/.

ConferenceXP (2007) *Microsoft Research Conference XP Project*, viewed October 8, 2007, http://www.conferencexp.net/.

eBeam (2007) *eBeam – Interactive Whiteboard Technology*, viewed 8 October 2007, http://www.e-beam.com.

Eclipse (2007) *Eclipse Rich Client Platform*, viewed October 8, 2007, http://www.eclipse.org/home/categories/rcp.php.

Evdokiou, P., Thomas, B.T., & Vernik, R.J. (2004) 'Augmented Synchronised Planning Spaces', *The Ninth International Command and Control Research and Technology Symposium*, Copenhagen, Denmark.

Fielding, R. (2000) 'Architectural Styles and the Design of Network-based Software Architectures', PhD dissertation, School of Information and Computer Science, University Of California, Irvine.

Fielding, R. & Taylor, R. (2002) 'Principled design of the modern Web architecture', *ACM Transactions on Internet Technology (TOIT)*, Volume 2, Issue 2, pages 115-150.

Gelernter, D. (1985). 'Generative communication in linda', *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H.F. (2003), 'SOAP Version 1.2 Part 1: Messaging Framework', World Wide Web Consortium, Boston, USA, 2003.

HxI (2007) *HxI Braccetto*, viewed October 8, 2007, http://www.hxi.org.au/.

JCR (2007) *Java Specification Requests – JCR 170*, viewed October 8, 2007, http://jcp.org/en/jsr/detail?id=170.

Johanson, B., Hutchins, G., Winograd, T. & Stone, M. (2002) 'PointRight: Experience with Flexible Input Redirection in Interactive Workspaces', *UIST-2002: Proceedings of User Interface Software & Technology*.

Johanson, B., Ponnekanti, S., Kiciman, E., Sengupta. C., Fox, A. (2001) 'System Support for Interactive Workspaces', Unpublished, available at http://graphics.stanford.edu/papers/iwork-sosp18/.

Knopflerfish (2007), *Knopflerfish OSGi*, viewed October 8, 2007, http://www.knopflerfish.org/.

Segall, B., Arnold, D., Boot, J., Henderson, M., & Phelps, T. (2000) 'Content based routing with Elvin4', *Proc. AUUG2K*, Canberra, Australia.

Sticker (2007) *Sticker home page*, viewed October 8 2007, http://tickertape.org/projects/sticker/.

Synergy (2007) *Synergy project*, viewed October 8, 2007, http://synergy2.sourceforge.net/.

Tanenbaum, A. & van Renesse, R. (1988) 'A critique of the remote procedure call paradigm', *In R. Speth, editor, Proceedings of the EUTECO 88 Conference*, pages 775-783, Vienna, Austria, April 1988. Elsevier Science Publishers B. V. (North-Holland).

Mimio (2007), *Interactive whiteboard, virtual whiteboard, whiteboards, Sanford Brands – mimio*, viewed 8 October 2007, http://www.mimio.com.

OSGi (2007), *OSGi Alliance*, viewed October 8, 2007, http://www.osgi.org/.

Vernik, R., Blackburn, T. & Bright, D. (2003) 'Extending Interactive Intelligent Workspace Architectures with Enterprise Services', *Proceedings of Evolve2003, Enterprise Information Integration*, Sydney, Australia.

Vernik, R.J., Johnson, S., Bright, D. & Vernik, M. (2004) 'Using Workspace Simulation to Support the Evaluation of LiveSpaces for Synchronised Planning Activities', *SimTecT 2004*, Canberra, Australia.

Waldo, J. (1999) 'The Jini Architecture for Network-centric Computing', *Communications of the ACM*, pages 76-82, July 1999.

Waldo, J., Wyant, G., Wollrath, A., & Kendall, S. (1994) 'A note on distributed computing', *Tech. Rep. SMLI TR-94-29*, Sun Microsystems Laboratories, Inc., Nov. 1994.

X10 (2007) *About X10*, viewed October 8 2007, http://www.smarthome.com/about_x10.html.