IET Publishing
The Institution of Engineering and Technology
ENGINEERING & TECHNOLOGY REFERENCE

Reference Article
1st published in March 2016
doi: 10.1049/etr.2015.0053
ISSN 2056-4007
www.ietdl.org

Untangling your Threads – a novel cloud computing application

Dr. John Yardley JPY Ltd., Surbiton, Surrey, UK
David Fox JPY Ltd., Surbiton, Surrey, UK
Thomas Michel JPY Ltd., Surbiton, Surrey, UK Faculty of Science, Engineering and Computing, Kingston University, London, UK
Dr. Gordon Hunter Faculty of Science, Engineering and Computing, Kingston University, London, UK

Abstract

Threads is a Cloud-based software service the authors describe as a *message hub.* It allows an organisation to store, search and share all its digital messages – emails and phone calls – to improve collaboration and productivity and to extract otherwise hidden information. Information overload and privacy concerns have conspired to suffocate the attempts of many firms to share their own data internally. Employees have responded by treating their company mail server as a private file server, something no mail server was ever designed to be. *Threads* addresses these issues at source by providing a framework where large amounts of data can be shared with confidence and searched, they believe, more easily than with individual private email accounts. *Threads* achieves this transparently to the user and requires no changes in working practices. It uses database de-duplication, speech and speaker recognition, artificial intelligence and a raft of human factors ideas to overcome the obstacles to data sharing. This study examines the background to *Threads,* its technology, and the work in progress. By way of example, it discusses the *Threads* Enron Database. *Threads,* they believe, is a unique and technically novel service which they present here as case-study of an application well-suited to Cloud implementation.

Introduction

As its name implies, *Threads* is designed to create coherent threads from messages aggregated across a whole company. A communication dialogue between a company and a third party often involves several employees from both sides, each using different media. Viewing that dialogue as a single thread makes the sequence of events much clearer to all concerned [1].

Sharing all of an organisation's messages by means of a single database presents many challenges. However, convincing the organisation's employees that they may benefit by extracting information easily, safely and with confidence can present many more.

So, why would an organisation want to do it? The answer is primarily because there is a massive amount of valuable company information that is either locked up or lost in private email accounts and telephone calls. Even while respecting privacy, it is possible to see trends and threats that no individual might ever be aware of. However, if users can be encouraged to share messages with colleagues, this adds even more value. It not only helps efficiency and collaboration, it encourages employees to be more responsible, open and cooperative.

That is all good for an organisation, but employees might ask why *should* they share their messages. Like everything that ever became 'open', it is more a question of why *not* share messages. For many companies and organisations, only a small percentage of the employees' communications are confidential, and the gains made in sharing non-confidential messages vastly outweigh the disadvantages of keeping them private. The biggest gain is transparency. A message sent from a user who keeps their mail private is often different in tone and content from one sent by a user who knows it will be shared. Furthermore, users get to see the bigger picture of a project or the company's business with a particular client.

Threads was designed under the mantra 'keep it simple'. Knowing message sharing might meet some resistance, it was important not to compound that with a difficult user experience. The aim was that *Threads* users would require no training nor documentation, nor change their working practices in any way. From the

moment the user logged in, its operation was intended to be obvious. This imposed some strict rules on the way data was presented and, as always, simplicity for the user involves complexity for the designer (compare the automatic gearbox for motor vehicles).

How did Threads evolve?

The *Threads* project evolved from the needs of a small company to share information whose representation had, over a period of 20 years, changed from physical to digital [2]. As such, the storage medium moved from shared physical access – filing cabinets – to private digital access – personal email accounts. While the digital form was easier to store and search, it often ended up locked in private user files. This applied primarily to company email, but also extended to documents (e.g. invoices, quotations etc.) that were not confidential and routinely needed to be shared amongst employees. As email became the de-facto currency of communication, where most documents were shared as email attachments, sharing the email became the key to re-establishing the collaboration ethos. While various procedures (e.g. using shared email folders) were put into place to allow the sharing of email, none was really successful. Apart from the fact that these required strict discipline on the handling of email, they often fell down when the user had several email accounts or was working from remote locations. Special purpose software – such as bespoke email clients – were universally shunned, and rightly so [3]. Gains made through digitisation were often lost through unnecessary privacy. Finally, but not least, staff were using their email server as a file server, something most email servers do badly.

Threads was written as web application driven from a database which ingested all company messages. As such, it could be used anywhere and did not require users to abandon their existing software – hence staff could continue to use their preferred email client and not change their working practices. *Threads* was designed as a collaboration tool, not as a replacement communication infrastructure.

While *Threads* was an excellent management and compliance tool, to promote effective staff collaboration, it demanded a paradigm shift back to the days where the default case was to share information. After some initial reluctance, staff soon realised that, with the appropriate controls in place, sharing could be liberating and few wished to return to the possessive 'old days'. Staff simply needed to review their perceived needs for privacy.

Having overcome this, employees needed to see real benefits before they would subscribe to using *Threads* given their perceived reduction in privacy. They immediately found this because, with *Threads*, they could find things they previously could not and, just as importantly, they were able to devolve responsibility when it was useful.

When digital telephony was introduced within the company [4] using an open source product called Asterisk [5], it seemed natural to include telephone calls – or recordings of them – within the same *Threads* message handling framework. An early design decision to unify the data structures for all types of message was fully vindicated. Once phone calls could be seen in the same context as text-based messages – and vice-versa – the true power of message sharing really became apparent. In the authors' case, the integration of phone calls into *Threads* was never motivated by legal compliance or disclosure issues. Indeed, early on, the decision was taken that calls would only ever be shared internally. What prompted this initially was to provide the knowledge that certain phone calls had occurred within the 'thread' of a project – without regard to the actual content of these calls. With the company's conversion to IP telephony, and the availability of call recordings in digital format, it seemed an obvious next step to include these within our *Threads* system. Once they were freely accessible to all staff, users realised they no longer needed to rely on scribbled notes or their recollection of a call. With *Threads*, they could find a call much as they would an email, listen to it as many times as they wished or pass the 'link' to relevant colleagues.

The pre-cursor to *Threads*, a system we called Mail Robot X (MRX), was originally implemented as a local-area network (LAN)-based server project. Once it was decided to commercialise MRX, it was re-engineered as a Cloud-based system which we now call *Threads*.

This article looks at some of the design criteria that evolved to meet the needs of a workable message sharing system. It goes onto discuss some of the implications of handling speech and the reasons why *Threads* was ideally suited to implementation in the Cloud. It then moves onto describe our recent work in progress: namely, integrating artificial intelligence, automatic speech recognition (ASR) and speaker recognition technologies into *Threads* to make it an even more powerful and useful tool to its users.

Eng. Technol. Ref., pp. 1–11
doi: 10.1049/etr.2015.0053

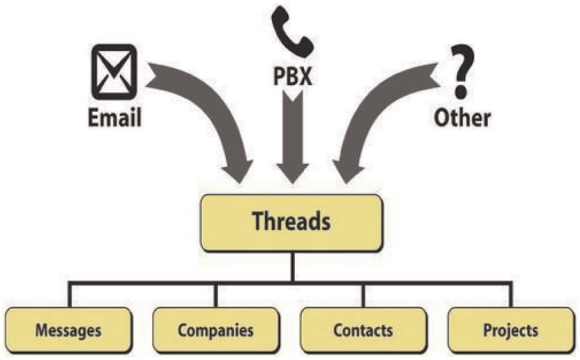

Fig. 1 *Simplified view of the Threads schema. PBX stands for 'Private Branch Exchange' and represents a source of telephone calls from the network*

Threads database

The starting point for *Threads* is the *Threads* database. The database contains many tables, but from the user's perspective, the user sees just four (Fig. 1): *Messages*, *Companies*, *Contacts* and *Projects*.

Since every message has to be stored, the *messages table* is a fundamental element of the *Threads* database. In principle, *Threads* can be useful simply for storing messages.

Messages are *unified*, in the sense that each message may be an email or (a recording of) a phone call but, depending on sources available, could be any type of digital message such as a simple message service (SMS) or *Skype* call.

The main key to classifying messages is to know *who* are the participating parties – i.e. who initiates the message and who else is involved. This gives rise to the *companies table* and the *contacts table* to define the contacts' addresses in the messages and the companies for which they work. (We use the term *companies* throughout this article to denote any organisation).

A subsidiary key to classifying messages is to know *what* they are about – namely, their information content. Where this information can be combined for a class of messages, it is stored in a table of *projects* (Again, the term *project* is used throughout this paper to indicate a collection of messages all relating to some common theme).

Except when inspecting an individual table entry – for example, reading a message or viewing an address – all the *Threads* user ever sees is a screen displaying one of these four tables (Fig. 2).

Threads keeps messages, contacts, companies and projects permanently related. So a search for a list of messages will instantaneously reveal the companies, contacts and projects that are involved in those messages. This is an important feature of *Threads* and adds much to its usability.

Threads dataflow

We define the *Threads* 'subscriber' as the organisation running the *Threads* service and 'users' as those people authorised by the subscriber to access *Threads*.

A new *Threads* subscriber will begin by defining a list of companies and contacts with which they normally deal. For subscribers with no existing contact list (e.g. a brand new company), this could be created by directly typing in company and contact details into the *Threads* user interface (UI). For companies with pre-existing contacts, lists can be imported from a file. Where the subscriber has an existing customer relationship management (CRM) system, *Threads* provides an application programmer interface to automate the synchronisation of the CRM system with *Threads*.

A subset of the contacts will be the 'authorised' *Threads users*. These are people with credentials allowing them to log into *Threads* and use it to view information. If *Threads* is used as a monitoring/compliance tool, there may be very few users. If, as we hope, *Threads* is used for collaborative working, then many employees of the subscriber may be authorised users.

To be of any practical use, *Threads* has to *ingest* messages. These may be emails, digitised [voice over IP (VoIP)] phone calls or other types of messages.

Message ingestion: In the case of emails, ingestion can be achieved by creating a *Threads* email account on any convenient server, from which *Threads* can periodically collect emails for ingestion. This *Threads* email account can be fed manually (e.g. by drag-and-drop) or by using simple rules that are coded and run on a user's email client or the organisation's email server. This means that there is no need to provide *Threads* with credentials to any private email accounts. Also, there is no limit to the number or type of accounts that might feed *Threads*.

New subscribers would be expected to begin by depositing email manually (i.e. using a drag-and-drop approach) and once they have built up confidence that

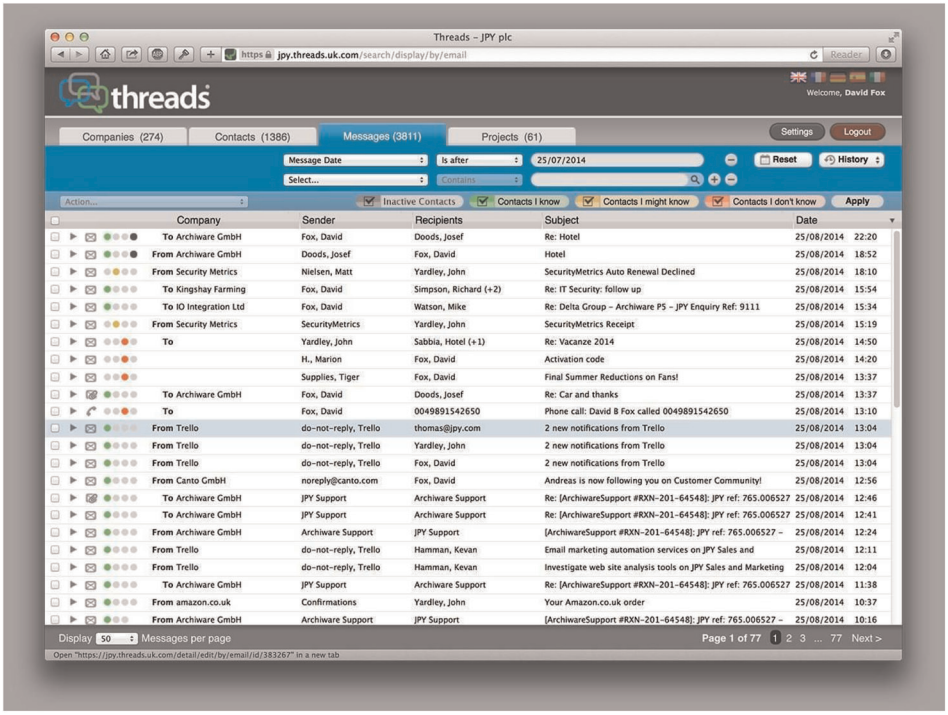

Fig. 2 *Typical web-view of the Threads message interface, showing both email messages and recorded phone messages. The 'traffic light' system of marking messages green, amber or red is explained in 'Orphans, 'contacts I know', 'contacts I might know' and 'contacts I do not know' section below*

Threads is sharing only intended messages, they invoke automatic processes to ingest messages.

In the case of (VoIP) phone calls, these may be ingested from a local or remote/hosted private branch exchange (PBX) database, by monitoring the subscriber's LAN for session initiation protocol traffic.

Though other types of messages such as SMS (SMS or 'texts') can be ingested, there is the obvious requirement that *Threads* can access them. This, as well as mobile phone calls, could be done either by the mobile phone service provider or with a *Threads* mobile 'app' to intercept appropriate traffic.

If users are to be encouraged to share messages, they need to feel confident that they are not inadvertently sharing their private messages. Any hope that employees can be prevented from using their company mail account for private messages is likely to be unrealistic.

Thus, rather than telling *Threads* what messages *not to* share, *Threads* adopts the approach of users telling *Threads* what messages it *can* share. This is achieved primarily with a database of company contacts. If a contact is in the *Threads* database and not flagged as private, messages exchanged with that contact are to be shared. If contacts are not in the database, then messages between the contact and the user are also kept private. There are many mechanisms to make company mail confidential – but the general rule is 'do not store personal contacts in the *Threads* contact database'. Staff can easily deal with such matters by using private address books to manage their private contacts.

Previous experience with CRM systems and contact databases revealed shortcomings we wanted to avoid with *Threads*. One frequent issue was with contacts who worked for several companies – either simultaneously or sequentially. When a contact leaves a company, the contact details cannot simply be deleted – the messages sent to and received from that contact still exist and may be relevant [6]. Users still need to know the details of the contact even if he/she no longer works for the company. Similarly, a contact may work for several companies or several different branches of the same company. Many CRM systems deal with this by cloning contacts, one high-profile example being SalesForce [7].

Threads takes a different approach by only ever storing a single contact record for each 'contact entity'. It then relates this contact record to one or

doi: 10.1049/etr.2015.0053

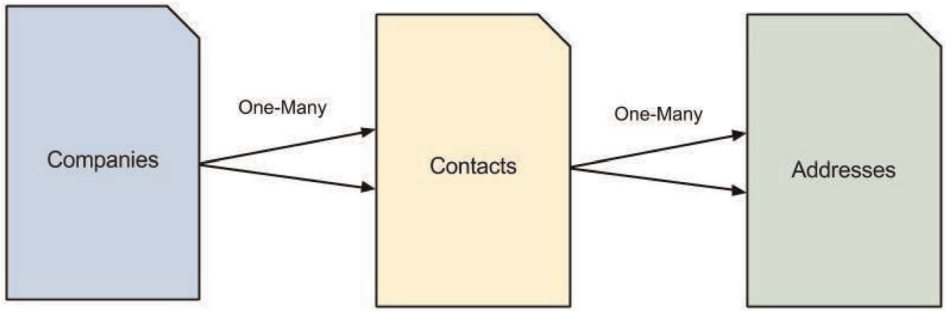

Fig. 3 *Conventional relationship between company and contact*

more companies by means of a *channel*. As such, it de-duplicates contact information.

Threads subscribers are not prevented from creating multiple contacts or companies, but it is preferable to create multiple *channels* for a single contact or company. The contact channel is another special feature of *Threads*.

Channel: The *contact channel* essentially establishes the relationship between the company and the contact for communications purposes. For each channel, there is a channel *type* and a channel *value*.

If Fred works for Acme Tools, then he might have a 'work email' type channel with a value 'fred@acme.com'. He might also have a 'mobile phone' type channel with a value of '07768-127656'.

Some channels might only be applicable to the company branch as a whole, for example, a switchboard number or a generic email address such as 'info@acme.com'. These we call *company channels*. Fig. 3 shows the traditional hierarchical relationship between companies, contacts and addresses (email, phone etc.). Fig. 4 shows the (simplified) *Threads* schema in which companies are related to contacts via channels. Rather than an address being seen as an attribute of a contact, a channel describes the way in which a company and contact are related.

Unifying emails, phone calls and other message types

Internet standards govern the detailed format of an email message and cover structures such as addresses, timestamps, attachments etc. [8]. However, for the purposes of our discussion we are only interested that an email comprises a list of email addresses (including the sender and recipients) and the message itself (that may contain attachments). From the email addresses, *Threads* can deduce the names of the contacts by looking up the contacts related to the channel value (i.e. email address).

In a VoIP [9] phone system, it is possible to trace each call either from a database maintained by a local or remote/hosted PBX [5] or, more generally, from 'sniffing' the relevant packets on the subscriber's LAN. Such is the importance of collecting phone calls independently of the PBX that we have developed an appliance specifically for the purpose.

A phone call can be treated much the same as an email except that the channel type will be different – i.e. it will be of the 'phone call' type instead of the 'email' type – and the value will be a phone number rather than an

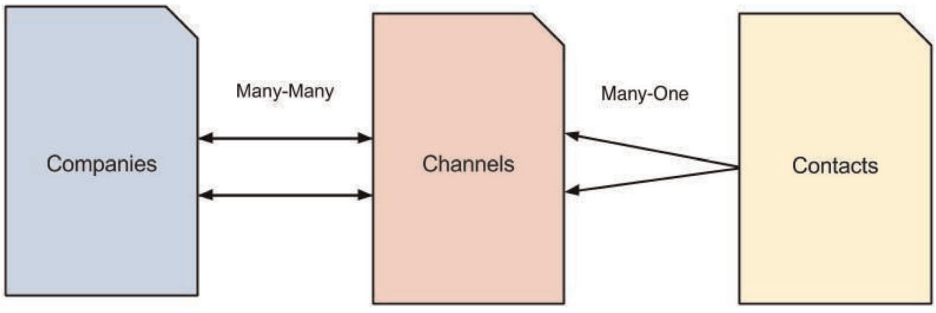

Fig. 4 *Threads 'channel' relationship between company and contact*

email address. In general, there will only ever be one sender (or caller) and one recipient (or person called). In the case where a call is transferred from one employee to another, then this can be treated as having multiple recipients, but relatively simple processing of metadata will reveal whether one or more of the 'recipients' was involved in the main dialogue.

The 'body' of a phone call is a digitised recording of a conversation, whereas the body of an email is a text string. Aside from these differences, a phone call may be treated almost identically to an email. The addressing does of course differ. An email address has a different syntax to a telephone number, but aside from this they have similar attributes. However, searching or indexing a text string (such as an email) is much more straightforward than doing the same tasks with a phone call, and we will discuss these in more detail later in this study.

Similar metaphors can be defined for other types of messages such as *Skype* calls and *Text* (SMS) messages.

Orphans, 'contacts I know', 'contacts I might know' and 'contacts I do not know'

Emails can, and often do, come from or go to email addresses not contained in company or contact channel tables. Similarly, phone calls may be exchanged using numbers not existing in channel tables.

Where there is no related company, *Threads* creates 'unrelated' contact records. *Threads* designates these contact records 'orphans' – since they have no parent company – or in the parlance of the *Threads* UI, 'contacts I do not know'. Emails containing orphans are marked as such and if the sender or recipient(s) are orphans, then the mail is only visible to the *Threads* user that sent or received the message. If the domain (e.g. jpy.com) of an email address matches an existing company domain in the database, then these contacts are termed 'contacts I might know'. The same happens for phone calls from a company switchboard number where the specific contact is unknown. Contacts that can be uniquely identified via an email address or a personal telephone number are termed 'contacts I know'. *Threads* uses a 'traffic light' system to mark each message as containing 'contacts I know' (●), 'contacts I might know' (●) and 'contacts I do not know' (●) (Fig. 5). This makes it very easy to filter mail according to contact type, the most obvious benefit of which is to hide unsolicited mail, spam and personal mail.

Note: We also use an optional classification 'inactive contacts' (●) to show that a message contains contacts that are currently inactive (e.g. left the company).

Projects, keywords and speech processing within *Threads*

The classification of messages using addressing information (i.e. to/from/cc) is easily defined and obvious to the user. It is also intuitively easy to search messages according to participants. However, the classification of messages by content is significantly more complex and deciding the right terms to yield a small list of relevant message can be difficult – all the more so given the diversity of contacts involved.

To make this process easier, *Threads* uses the concept of *projects* – which may be seen as folders which contain all the messages that relate to a particular project. It is generally easier to *browse* projects than to *search* for terms whose distribution is unknown across the complete set of messages.

Ultimately, only a human can decide whether or not a message belongs to a particular project, though we hope eventually that *Threads* will pass the Turing test

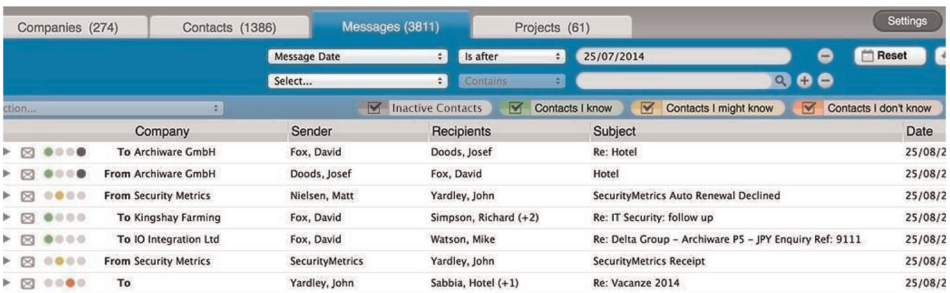

Fig. 5 *Threads UI with contact 'traffic lights'*

doi: 10.1049/etr.2015.0053

for this task and make it impossible for a user to decide if the classification was made by *Threads* or a human.

In the meantime, we are developing ways to make the selection process much easier for the human and we devote the rest of this article to describing some developments currently in the pipeline.

Projects and keywords

Threads allows users to create projects and a hierarchy of sub-projects. For example, a project called 'The M4 extension' might have sub-projects called 'Tarmac contractors' and 'Health and Safety'.

When a user views a message, he/she has the option of allocating it to one or more projects. Once allocated, *Threads* extracts information from the message and includes it in the project template. This information includes a list of *keywords* – or words with a high information value. As noted by Shannon in his seminal work on Information Theory [10], information content is inversely related to a priori likelihood of occurrence. Highly common, and hence probable a priori, words such as 'and', 'but' or 'is' have very low information value, whereas much less common words such as 'demography' or 'concurrency' have much higher information value.

When offering the list of available projects with which to associate the message, *Threads* ranks them in order of best fit with the message of current interest – the objective being that the project with the best fit appears highest on the list. As more messages are added to the project, the more representative of a generic message relevant to that project the word statistics of the project become.

The crucial information that *Threads* is able to extract is context relevance. Instead of making decisions in isolation about how a message best fits a project, *Threads* can and does use a mass of context information from previous messages. For example, a particular user might never use the word 'spridget' and so, independent of other content, that user is less likely to be associated with projects involving spridgets. Conversely, another contact might make extensive use of the word 'spridget' in his/her messages, so those messages are most likely to be relevant to those projects where that word is common.

The details of this are beyond the scope of this article, but the interested reader is referred to a more complete discussion by the authors, including descriptions of experiments and their results in [11, 12].

Handling speech

Searching for text strings in an email or a set of emails is straightforward for a computer. Searching for a specific thing or things within phone conversations poses more of a challenge.

With *Threads*, we are investigating the use of two main areas of speech technology: automatic speech recognition (ASR) and automatic speaker identification. A readable introductory overview of the field of speech technology is given by Holmes and Holmes in [13].

Speech recognition: The process of converting a human speech waveform into digital text is known as ASR. The 'Holy Grail' of ASR is the speech dictaphone – a device that can, without training or subject knowledge, reliably produce an accurate digital text transcription from continuous human speech – just as human might. There are many ASR systems – both commercial and open source – available and users have had a wide variety of experiences with them – from finding them indispensable to unusable. Some examples of such ASR systems include Nuance's commercial product *Dragon Naturally Speaking* and Apple's *Siri*, and the open source product *Sphinx* from Carnegie Mellon University in the USA.

Though ASR is an area of particular in-house expertise [14], and a solid understanding of ASR is an essential prerequisite to the successful application of ASR, it was never the intention to develop ASR processes specifically for *Threads*. Instead, the aim is to use the best available ASR in class. No matter what the current performance of ASR systems, they will undoubtedly improve over time and as they do, we will be able to pass those improvements onto *Threads'* users straight away. It has been widely recognised that the limited bandwidth of the telephone channel and other issues such as noise and cross-talk make the problem of automatically recognising and transcribing speech more challenging over a telephone line than under more ideal conditions [15], and use of recordings of telephone calls rather than live calls may exacerbate these problems further. However, as we discuss below, the aim within *Threads* is not to produce an accurate verbatim transcript of a call, but rather to reliably identify its theme and to which other messages it is related.

However, the novel characteristics of *Threads* require radical changes to the criteria by which most ASR systems may be measured. For example:

(i) Full speech transcription is not a requirement. The objective is to allow phone calls to be searched – what might be called 'speech indexing' of the message. As discussed, rare things have high information value, but very common things tend to carry little information. Thus, search terms which use words with low information value are rarely of much help in locating messages (e.g. searching for the word "and"). Although not always the case, words with a high information value (keywords) are often acoustically complex and easier to isolate from a speech waveform. Hence, an ASR algorithm that renders poor speech transcription performance might still be useful for searching if it correctly identifies enough keywords (See also [11, 12, 16]).

(ii) Real-time operation is not a requirement. It may be days, weeks or years before a message is searched; hence, ASR algorithms can be applied that are not fast enough for real-time use or when computing power is cheap.

(iii) Costs can be adjustable. Already there are many ASR services available as remote resources, each with different strengths and different costs. Subscribers could choose between different algorithms according to financial resources.

(iv) Context. *Threads* makes available context information that would be totally absent in a 'dictaphone' situation. Context information includes knowledge of the speakers and the vocabularies they use, as well as the projects they are known to be working on.

Again, space prohibits a detailed discussion of our work in this field, but we are presenting this work to the speech centric communities. Fig. 6 shows the relationship of the various speech-related modules including the context-based language model. This work is being undertaken as a collaboration between JPY Ltd. and Kingston University.

Speaker identification: Automated identification of speakers through analysis of the acoustic profiles of their speech (as observed from recordings of their phone calls) is potentially fertile ground for extracting context information. This has been a field of serious research by various authors since the 1970s [17] and major advances have been made [18]. Though biometric applications of speaker recognition (e.g. for use in identifying, whereas someone should be granted access to some secure resource such as a bank account, on the basis of their voice profile alone) have met with limited success and application to date, since intra-speaker variation may be comparable with inter-speaker differences, in our situation, further contextual information is often available. In VoIP telephone systems, each party's speech in a two-way telephone conversation is exchanged on its own (logical) channel. Unlike an analogue phone call, this permits the speech pattern for each speaker to be isolated. By using dialing information (i.e. knowing what

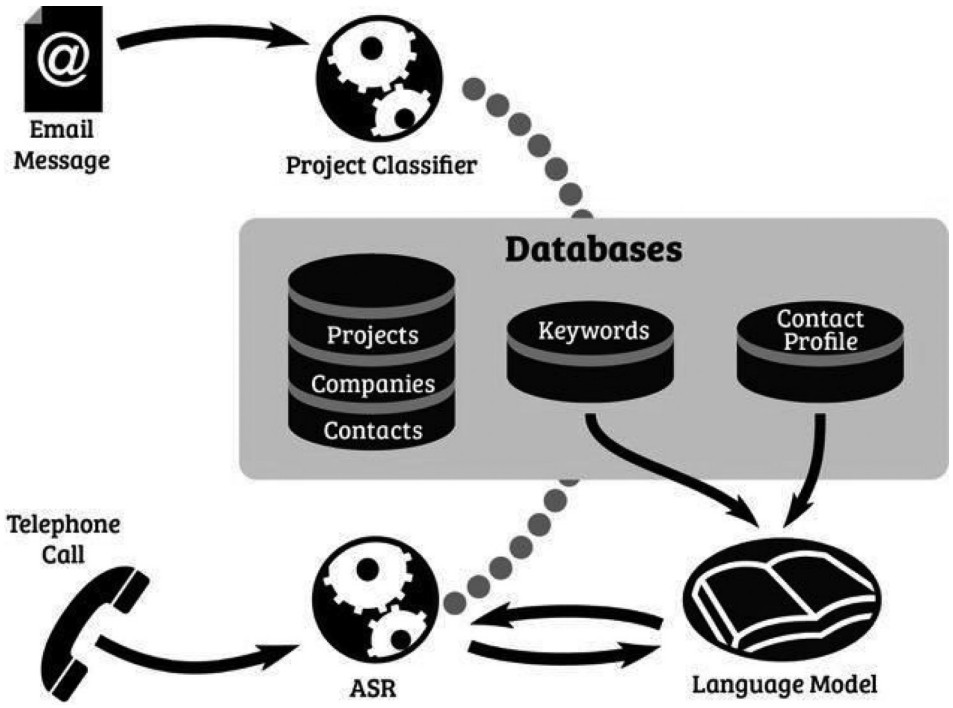

Fig. 6 *Schematic view of ASR and language model modules as incorporated in Threads*

Eng. Technol. Ref., pp. 1–11
doi: 10.1049/etr.2015.0053

company or individual made the call) or input from the user, it is possible to build an acoustic profile of the voice of each contact. The problem is no longer one of identifying a speaker from an extremely large set of possible people, but rather one of selecting the correct speaker from a small set of profiles. This could also be of assistance in the speech recognition process – if *who* is speaking can correctly be identified, then a specialised profile of word statistics (or 'language model') commonly used by that particular person could be used to assist in the ASR for keyword identification.

Automated speaker identification may be used in the following ways:

(i) To identify an unknown contact from a list of known contacts using a single switchboard number and a set of profiles for all the speakers associated with that number.
(ii) To establish a contact as unlikely to be known to *Threads,* on the basis of his/her speech profile being very different from every profile stored in the *Threads* database.
(iii) To establish that a caller is masquerading as a known contact with a very different speech profile.

As with speech recognition, it is not essential (except in cases where it is desirable that fraudulent callers are identified immediately) that this process happens in real-time. It can be deferred to a time when computing is cheap and/or call volume is low (e.g. at night). Similarly to the case of ASR, as better speaker models and algorithms evolve, they can be immediately adopted to give better performance without negatively impacting the user.

However, with the exception of a few highly specialised applications, speaker identification has not yet been widely adopted as a means of identifying a user by businesses. This suggests that there are still many challenges to be overcome in this field [19].

Cloud implementation

Originally, *Threads* was planned as a LAN-based (on-premise) server application. By the time the idea of commercialising *Threads* had formed, Cloud computing was gaining traction as a compelling platform. This was for the following reasons:

- The input/output rates and storage requirement for email and phone calls are relatively low per user – well within Internet bandwidth limits of anticipated subscriber companies.
- Deployment as a single, scalable service reduced the cost and effort of maintenance and upgrading of the service. It was far easier than local deployment on the subscriber's hardware.
- The Cloud was highly conducive to integration of third-party processing services such as signal/data analysis and remote/hosted PBX systems.
- The fixed costs were predictable and subscription to the service required no subscriber commitments.

If any application could be described as ideal for implementation on the Cloud, then it was *Threads*.

Threads is currently being developed using the Amazon Web Services (AWSs) Elastic Cloud Computing (EC2) platform [20]. The web application servers run behind a 'load balancer', allowing easy scaling by adding webserver instances on demand as requests seen by the load balancer increase. *Threads'* application servers run on Linux, using Apache and PHP. They connect to a Windows server hosting a 'Microsoft SQL Server' database, again running within the EC2 environment. Email ingestion from clients' servers runs on a separate Linux server, again running PHP and writing to the SQL Server database as email and other messages are added. Email attachments and phone call recordings are stored in AWS S3 storage, for security and ease of access.

At the time the decision was first made to commercialise Threads as a Cloud-based service, many of the functions that would eventually be necessary for a commercial service were untested or, mostly, not available. These include:

- Solid state disk storage on our instances for increased performance.
- Offline storage in the cloud, for archiving old email at lower storage costs.
- Virtual private cloud, for secure isolated IP subnets.
- Higher performance, cheaper and instance types.
- Custom AWS Linux distribution and update repositories.
- 'CloudFormation' for building our instance software stack (2011).
- EC2 for Windows (2008).
- Elastic Public IP's (2008).
- S3 storage (2008).

Fortunately, these services have since been released and deployed, but at the outset, adoption of the Cloud was very much an act of faith.

Threads Enron database [21]

A major part of the future development of *Threads* involves testing our various threading strategies. This requires real operational data since simulating large volumes of messages can produce misleading results.

At the outset of the project, the only data available was our own (JPY Ltd.). Our current corpus comprises over 500,000 emails and approximately 10,000 digitised phone calls. For confidentially reasons, few outside companies were found that were prepared to share their mail corpora and even fewer had recorded their telephone calls. Obtaining other operational data for experiment and test was nevertheless seen as an important part of the project. Algorithms which perform well on one set of data may not work well on a different dataset – as was the case in the infamous early application of neural networks by the U.S. Military to decide whether a given photograph did or did not contain a concealed tank [22].

To remedy this, we set about ingesting the entire public-domain corpus of Enron emails and phone calls. The collapse and bankruptcy of the mighty US Enron Corporation in 2001 was one of the world's largest corporate scandals [23], leading to major fraud and non-compliance trial. One consequence of the investigation was the decision of the US Department of Justice to make a large proportion of Enron's data – including some 250,000 emails, relating to 350 projects, involving around 28,000 contacts from 1500 companies, plus 76 recordings (with transcriptions) of telephone calls – available in the public domain [24].

The ingestion of this data was a time-consuming and complex operation since the emails were available in a bespoke database which distorted much of the original metadata. The phone calls were published in a number of disparate locations, and again some metadata was not available. The fact that we were able, by various means, to obtain court transcriptions of phone calls provided an invaluable dimension that was unavailable from our own corpus.

Using *Threads* we were able to reconstitute the Enron corpus into a searchable form and restore a significant amount of lost information. This has been published publicly as the TED [21] and, as such, has attracted many users – academic, journalistic and commercial. The reader is referred to the TED web page for more background to Enron and to TED. Using both the JPY dataset (described above) and the Enron corpus, we were able to perform experiments, both on text data and on transcribed telephone calls for which a high-quality transcription was available, on keyword identification and the use of keywords for message classification using data from two completely independent sources. This enabled us to test the reliability and robustness of our methods and ensure that the algorithms were not just suitable for use on one particular dataset. Full details of these experiments, and their results, are given in [12, 25].

Conclusion

We hope this article has provided an interesting overview of what we believe is a novel Cloud application relevant to companies and organisations of any size and in any sector.

While much of the internal design of *Threads* is contained within various patent applications [26], we trust this article gives a broad overview of the technologies and methods employed and under development.

Applications for *Threads* include management reporting, legal compliance, data security, backup, archive and collaborative working. It is especially useful in project-oriented professions such as law, architecture and engineering. What is particularly exciting is *Threads'* potential for extracting otherwise untapped knowledge and strategic help from messages already in a subscriber's possession – that is, *Threads'* 'Big Data' potential.

The use of such a system raises many issues and it is yet to be established whether some larger organisations are ready to share internally their non-confidential messages. However, as a management reporting and discovery tool, we feel it has much to commend it.

The *Threads* framework is a starting point and the information that can be extracted by use of complex technologies such as speech recognition and artificial intelligence is limitless. Complex as these technologies are, they are used to make UI as intuitive as possible. We have tried to keep to the path that, above all, *Threads* must be simple to use and understand.

We would be grateful for any feedback on *Threads* and would welcome hearing from any organisation

doi: 10.1049/etr.2015.0053

interested in trialling the system in their own business environment.

Acknowledgements

The authors' work on speech processing, keyword analysis and various other aspects of the *Threads* project have been undertaken as a collaboration between JPY Ltd. and Kingston University - supported by a Knowledge Transfer Partnership (KTP) award from the UK government agency *Innovate U.K.* (UKinnovate.org). We also wish to acknowledge the contribution of software engineers, Karen Clarke and Narinder Chandi to the development of *Threads* project.

REFERENCES

[1] JPY plc.: 'Threads website', 2014. Available at http://www.threads.uk.com
[2] Chowdhury, G.: 'Introduction to modern information retrieval' (Facet Publishing, London, 2010, 3rd edn.)
[3] Schuff, D., Turetke, O., Croson, D.: 'Managing email overload: solutions and future challenges', *IEEE Comput. Soc.*, 2007, **40**, (2), pp. 31–36
[4] Yardley, J.: 'Adopting an open source VoIP phone system'. Available at http://www.jpy.com/support/articles/adopting-open-source-voip, 2010
[5] Imran, A., Mohammed, A.Q., Khan, M.: 'Asterisk VoIP private branch exchange'. Int. Multimedia, Signal Processing and Communication Technologies. IMPACT'09, IEEE, New York, NY, USA, 2009, pp. 217–220
[6] Lazarov, V.: 'Comparison of different implementations of multi-tenant databases', in IEEE Components, Packaging, & Manufacturing Technology Society (IEEE CPMT-Taipei) (Eds.): 'The CRM database schema' (Technische Universitat Munchen, 2007), pp. 12–13
[7] Salesforce Inc.: 'Salesforce Website', 2016. Available at https://www.salesforce.com. For more information on cloning, perform a web search on 'Salesforce contact cloning'
[8] Internet Engineering Task Force RFC memoranda 2046, 2047, 4288, 4289 and 2049
[9] Davidson, J.: 'Voice over IP fundamentals' (Cisco Press, Indianapolis, IN, USA, 2006)
[10] Shannon, C.E.: 'A mathematical theory of communication', *Bell Syst. Tech. J.*, 1948, **27**, (3), pp. 379–423
[11] Hunter, G., Denholm-Price, J., Michel, T., *et al*.: 'Keeping your threads untangled, an intelligent system for semi-automatically organising corporate messages by content'. Proc. 11th Int. Conf. on Intelligent Environments, 2015
[12] Hunter, G., Denholm-Price, J., Michel, T., *et al*.: 'Progress towards a smarter office via a novel intelligent system for message organisation'. Proceedings Fourth Int. Workshop on Smart Offices and Other Workplaces (SOOW), Prague Amsterdam, 2015
[13] Holmes, J., Holmes, W.: 'Speech synthesis & recognition' (Taylor & Francis, London, 2001, 2nd edn.)
[14] Yardley, J.: 'Word identification in speech by phonetic analysis'. PhD thesis, University of Essex, 1981
[15] Gauvain, J.L., Lamel, L., Schwenk, H., *et al*.: 'Conversational telephone speech recognition'. Proc. IEEE Int. Conf. on Acoustics, Speech & Signal Processing (ICASSP), 1988
[16] Yardley, J., Hunter, G., Denholm-Price, J., *et al*.: 'Using ASR to get data out, not in'. UKSpeech Conf., University of East Anglia, Norwich, UK, July 2015
[17] Atal, B.S.: 'Automatic recognition of speakers from their voices', *Proc. IEEE*, 1976, **64**, p. 460
[18] Reynolds, D.A.: 'Automatic speaker recognition using Gaussian mixture speaker models', *Linc. Lab. J. (M.I.T)*, 1995, **8**, (2), pp. 173–192
[19] Singh, N., Khan, R.A., Shree, R.: 'Applications of speaker recognition', *Procedia Eng.*, 2012, **38**, pp. 3122–3126
[20] Amazon Inc.: *Amazon Web Services*. Available at http://www.aws.amazon.com
[21] JPY Ltd.: Threads Enron Database. Available at http://www.threads.uk.com/threads-enron-database
[22] Fraser, N.: 'Neural Network Follies', 1998. Available at https://www.neil.fraser.name/writing/tank/
[23] Smith, R., Emshwiller, J.: '24 days: how two wall street journal reporters uncovered the lies that destroyed faith in corporate America' (Harper Collins, New York, 2011)
[24] Klimt, B., Yang, Y.: 'The enron corpus: a new dataset for email classification research'. Proc. of European Conf. on Machine Learning (ECML), Pisa, Italy, 2004
[25] Hunter, G., Denholm-Price, J., Yardley, J., *et al*.: 'The contribution of automatic speech recognition for keywords to assist in the integrated organisation of digital messages', *Proc. Inst. Acoust.*, 2015, **37**, (2), pp. 260–267
[26] JPY Ltd.: 'Digital messaging system'. UK Patent Application No. 1408302.6, 2015, International (PCT) Patent Application No. PCT/GB2015/050580, 2015
[27] Majewski, W., Myslecki, W.: 'Application of automatic speech recognition to evaluation of speech transmission quality in analog communication systems', *J. Acoust. Soc. Am.*, 1999, **105**, (2), p. 976