CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (2013)

SPECIAL ISSUE PAPER

Approaching the Internet of things (IoT): a modelling, analysis and abstraction framework

Ahsan Ikram[1], Ashiq Anjum[2,*,†], Richard Hill[2], Nick Antonopoulos[2], Lu Liu[2] and Stelios Sotiriadis[2]

[1]*Bristol Institute of Technology, UWE, Bristol, UK*
[2]*School of Computing and Mathematics, University of Derby, Derby, UK*

SUMMARY

The evolution of communication protocols, sensory hardware, mobile and pervasive devices, alongside social and cyber-physical networks, has made the Internet of things (IoT) an interesting concept with inherent complexities as it is realised. Such complexities range from addressing mechanisms to information management and from communication protocols to presentation and interaction within the IoT. Although existing Internet and communication models can be extended to provide the basis for realising IoT, they may not be sufficiently capable to handle the new paradigms that IoT introduces, such as social communities, smart spaces, privacy and personalisation of devices and information, modelling and reasoning. With interaction models in IoT moving from the orthodox service consumption model, towards an interactive conversational model, nature-inspired computational models appear to be candidate representations. Specifically, this research contests that the reactive and interactive nature of IoT makes chemical reaction-inspired approaches particularly well suited to such requirements. This paper presents a chemical reaction-inspired computational model using the concepts of graphs and reflection, which attempts to address the complexities associated with the visualisation, modelling, interaction, analysis and abstraction of information in the IoT.

KEY WORDS: Internet of things; distributed systems; chemical computing; formal modelling

1. INTRODUCTION

The Internet of things (IoT) [1] is defined as uniquely addressable objects and their virtual representations, in an Internet-like structure. One of the major enablers of IoT is the plethora of information that will be generated by all the 'things' in IoT, and consequently, a major outcome of IoT will be enriched awareness, interaction and communication of an individual entity (or a community) with its surroundings, both virtual and physical. The IoT umbrella redefines boundaries between humans and devices, virtual and physical; the interactions need not be limited to humans accessing services or devices; they can be much more enriched and interactive [2, 3]. This enrichment of interaction, in turn, demands more detailed and comprehensive understanding of the context of a situation, event or interaction. It also demands presentation of that context and the awareness based on that context to the actors participating in the interaction. This takes computing interaction between entities from 'consumption of services' to the next level, that being 'interactive dialogue'. Moreover, with the success of virtual social networks, social interaction and communities have become a necessary

of any next generation computational framework. All these aspects of evolution of interaction, from man-using-machine to two networked entities having a dialogue, together with the existence of social groups and communities amongst these dialogues, make the communication and interaction model of IoT much closer to the nature-inspired communication and interaction models [3] that 'stigmergy' used in [2]. Similarly, as pervasive and ubiquitous devices merge the boundaries between real and virtual spaces, and claim equivalence to humans, nature-based computation models offer answers to the reasoning and modelling complexities of IoT. The concepts of habitats, groups, communities, languages, proximity, discovery and exploration now become even more relevant.

As highlighted in [4], a lot of research is in progress to develop architectures, frameworks, data models and communication protocols that can eventually realise the IoT concept, and there is active research in progress that investigates nature-inspired models for the realisation of IoT [5]. In this paper, we highlight the key challenges associated with realising IoT and present an end-to-end framework based on a combination of nature-inspired chemical computing and reflective middleware, to address the aforementioned challenges.

2. INTERNET OF THINGS: KEY CHALLENGES AND REQUIREMENTS

In this section, we highlight the core challenges and requirements that any IoT framework should be able to offer.

2.1. Abstraction and communication

In the IoT, entities with different capabilities and preferences, in terms of communication protocols, data models, battery life and so on, will be connected under the same network. Two devices may provide equivalent information yet utilising two completely different data models and/or units. Therefore, an abstraction layer is required to hide these complexities from the layers that enable interaction. These interaction models are moving on from service-based consumption (where a service was specifically written and designed for a specific identified purpose, network and/or device) towards an 'everything everywhere' model, and this requires a framework that separates the complexities of the information from the interaction model.

2.2. Adaptation and scalability

In a proposed IoT infrastructure, new entities will join the network every moment, where this increases the richness of the overall knowledge of the network. This requires a framework that is able to adapt to new members of the family via flexible interfaces, 'plug and play' architectures and/or standardised access methods. The framework and its data model should also be scalable to handle the growing nature of the information it handles.

2.3. Interaction and visualisation

Interaction and visualisation are the newest features that an IoT will enable. The users in IoT are no longer consumers, rather they are motivated to get involved and interact with the immense information and applications that an IoT offers, and they want to consume services they have designed and orchestrated. For example, users do not want their phones to be aware of their location anymore, but they do want to design how their phone will utilise their current location. These user-centric and user-defined interactions, along with the need to constantly access information, requires a visualisation layer atop the information layer, that not only provides access but also serves as a blackboard, or a 'playground' where queries can be explored.

2.4. Modelling and analysis

As the detail of the available context increases, so do the processes to analyse and store it. The decisions, actions and reactions in the proposed IoT environment will be dynamic, real time and frequent. Therefore, efficient underlying data models and reasoning methods will be required.

In this paper, we propose that a nature-inspired chemical computational model is well suited to all of the basic requirements of modelling and visualising context as information, and it also offers intuitive and natural visualisation and reasoning models. We propose that IoT is a large-scale extension of pervasive connectivity and context awareness, and therefore, the chemical computational model can be utilised to address IoT challenges.

3. A CHEMICAL METAPHOR

Chemical computing is a nature-inspired computing model that proposes computation on the concepts of chemical reactions, elements, bonds and solutions. Most of the existing chemical computing models are derived from γ (gamma) programming language [6]. Chemical computing separates information from the logic and supports scalability as new information and logic can be added to the system dynamically. The separation and other chemical characteristics of spontaneity and dynamic interaction enable multiple states of existence and more support for the management of finite but large and concurrent interaction computation.

Dittrich *et al.* [7] presented a comprehensive collection of chemistry-inspired computing approaches. Chemistry-inspired models have previously been used in workflow enactment research [8–10]. The chemical model has also been used to enable dynamic service composition [11]. J. P. Banatre *et al.* [12] suggested a chemical model for programming and modelling of self-organising systems. Tschudin and Yamamoto [13] applied a chemical execution model, Fraglets, to the implementation of communication protocols. Lin *et al.* [14] analysed modelling of parallel computation using the gamma chemical computation model [6]. Predominantly, the chemical metaphor has been adopted in research to address non-determinism, scalability and adaptability. Although modelling a context-aware response is analogous to enacting a workflow or composing a service, we explore a unified chemical model that enables modelling of both context and services. This makes it a novel approach because other approaches place an emphasis upon the composition of either context or services but not both.

4. CHEMISTRY FOR CONTEXT AWARENESS (C_2A)

In this section, we briefly describe C_2A and revisit some of its core concepts.

Context element (CE): CE is the basic unit of interaction. Each CE has a set of properties and interfaces that it uses to bond with other CEs.

Context periodic table (CPT): CPT is an organisation of CEs on a nature-based classification that puts each CE in one or more of six categories. The use of word 'periodic' is symbolic only in CPT.

Compound: Two or more CEs can combine to form higher order structures called compounds.

Bond: Two or more CE can come together and form a compound using various types of bonds that are part of C_2A.

Smart space (Ss): An Ss is a closed existence of various CEs and compounds; in terms of chemistry, an Ss is analogous to a solution. There are two types of Sss an entity or user can be part of: social smart space (SSs) and personal smart space (PSs). In the scope of this paper, an Ss can be seen as a model of connected things.

In the following section, we propose an end-to-end architectural framework based on the building blocks of C_2A that enables its realisation.

5. SYSTEM ARCHITECTURE TO SUPPORT REFLECTION

Reflection is a concept by which an entity can reason and review its state and then adapt its behaviour accordingly at runtime. Therefore, reflective models suit environments with finite but large set of

possible interactions, such as the IoT, by providing support for adaptation, awareness and reorganisation [1]. Structure and behaviour are considered two dimensions of reflection. Structural reflection is the concept where a programme can make changes, typically in terms of programming language data structure, methods and state. Behavioural reflection is a means by which an invoked method can be modified and tends to be more prevalent in middleware and application semantics. Logically, a reflective computing model is distributed into base and meta levels. In the context of a reflective middleware, the base level deals with the representation and interaction of applications and services with users. The meta level contains components and structures that enable reflection of behaviour on behalf of the base level requirements.

Licia Capra [15] proposed a reflective middleware, CARISMA, to support proactive and reactive modes of pervasive interaction. Other reflective middleware research includes PURPLE [16], ReMMoC [17], MobiPADS [18], Hydra [19], CAPIM [20] and COSAR [21]. Hydra is a context- based adaptive and reflective middleware approach that aims at integrating and optimising various wireless access technologies available to applications in a mobile device and wireless environment. Whereas the scope of the project is limited to wireless access technologies and a pre-defined policy-based adaptation, the concept of using an adaptive middleware is significantly related to this work. CAPIM and COSAR are two more context-based initiatives, where CAPIM provides an end-to-end framework for management of context and information and COSAR is aimed at complex activity recognition based on context; both projects however make use of ontologies for managing and reasoning information. Ontologies are seen as one of the core concepts that can be used to manage information whether it is context or IoT; however, ontologies focus on reliability and completeness of information and require structured data, which may not be the primary goal in a dynamic and rapidly changing environment such as IoT where quickness of response, partial reasoning and incomplete data may be often involved.

Tuple spaces were proposed as an implementation of the Linda computing model [14]. Conceptually, a tuple space can be described as a centralised set of information, with concurrent and synchronised access to a distributed system. By providing a shared space, tuples enable a system to be asynchronous, dynamic, shared and robust [22, 23]. Pervasive environments are characterised by users connected to an intermediate network component (such as gateways, routers, access points and base stations). Tuple spaces enhance scalability, synchronisation and dynamic interaction amongst pervasive users and devices. It is proposed in this article that the identification of context-based communities in a pervasive environment can be addressed by enabling tuple space models at the access gateway. Tuples (which are context-based Sss in this research) are gathered at these spaces and analysed. Moreover, tuple spaces are organised hierarchically such that the gateway spaces are child nodes of a central space. This hierarchical arrangement allows hierarchical abstraction, organisation and clustering of information contained in tuples (context) [14].

We thus propose that a hybrid architecture, which supports tuple space behaviour amongst different entities, will enable the bringing together of Sss from different users for a collective-shared representation and analysis. Moreover, adding support for reflective behaviour and interfaces for interaction, with the chemical model, will support the realisation of dynamic and spontaneous adaptation based upon chemical reactions. It also facilitates user participation in the process of enabling awareness by providing an intuitive meta interface.

Figure 1 shows the abstract overview of the proposed framework. The overall architecture works on the basis of the C_2A interaction model, such that, CE and Ss are units of interaction, context bonds, reactions (R) and two reflective middleware components are enablers of interaction and solutions are containers of interactions.

5.1. Architecture layers

Figure 2 shows the architectural layers where each layer encompasses a functional aspect. Ss interaction manager (SSIM) is the core component and is distributed in user and middleware planes. SSIM in the middleware layer is the central tuple space component of the architecture that is responsible for gathering all Sss; SSIM in the user plane is a lightweight reflective component that manages context and reactions for a single user. The following subsections define the functionality and composition of each layer illustrated in this architecture.

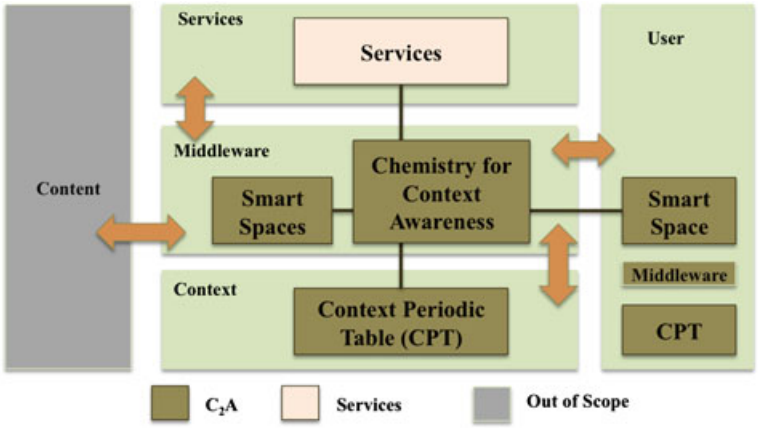

Figure 1. Architecture overview.

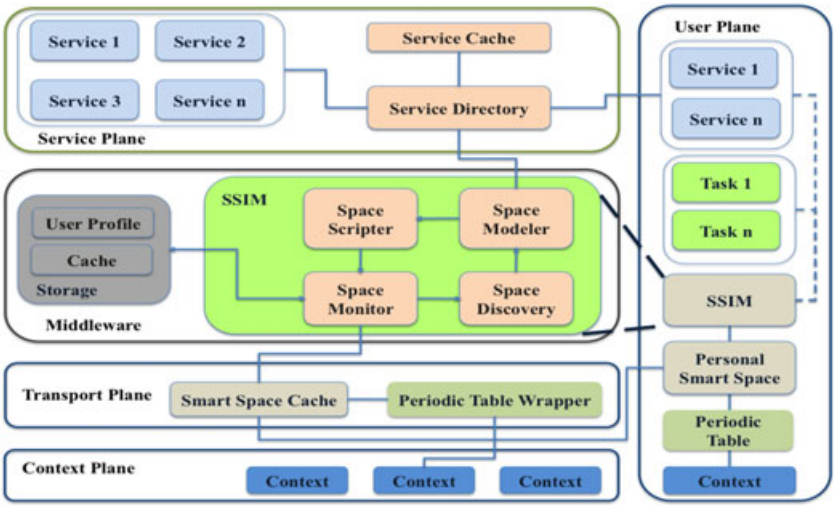

Figure 2. Architectural overview based on functional layers.

5.1.1. Context acquisition plane. Raw context is gathered at this layer. If the context received is in the form of a CE, it is passed onto the Ss cache that analyses it before passing it further. If the context received is in raw form, that is, in any form other than a CE, it is passed onto the CPT/CE wrapper component that generates a formal CE/CPT representation for the raw context, as shown in Figure 3. In Figure 3, the data structure is the entity that holds the schema to store information about the CE; 'handler' is the interface used to access the CE. 'Handler' is a system level platform specific service, for example, a Java thread.

5.1.2. Transport plane. A transport plane in a pervasive environment is the plane that manages user registration with the network via an access gateway. An access gateway is associated with a physical location and is therefore of vital importance in this architecture because spatial modelling and management of the middleware is performed at this layer. The component 'Ss cache' maintains an image of the PSs of entities and services registered with this gateway. This component enables realisation of physical proximity-based SSss.

5.1.3. User plane. The plane deals with the entities in the environment, which can be users, sensors, devices or a combination of any of the aforementioned list. In its composition, a user plane consists of two core components and three optional components. PSs and space manager are the core components, and services, tasks and context are optional. An entity may or may not host services and tasks and

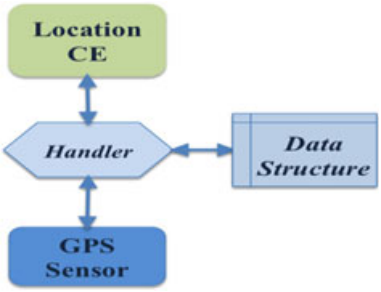

Figure 3. Context acquisition process.

provide context. SSIM is the component that enables reflection and user participation. PSs is the basic unit of interaction in the proposed framework. Ss generation is the main responsibility of the user plane. Figure 4 gives an overview of the process.

In the first phase, context is gathered from the users, devices and environment. The gathered context is raw and cannot be distinguished in terms of their roles in the Ss modelling. However, each context is gathered through a context handler, which is registered with the SSIM. CPT creates an entry for the CE on the basis of its *type*, *category* and *reliability*. Every user in the environment providing or consuming context owns an instance of the CPT. The following section explains the working and structure of an SSIM.

5.1.3.1. Smart space interaction manager. The SSIM is the reflective component of the architecture that is responsible for managing adaptation for PSs. Figure 5 shows the structure and composition of SSIM within the user plane. The SSIM is a reflective component and hence by the traditional definition of reflection has a base level and a meta level. The meta interface is distributed in two levels, meta and meta', where meta' is logically the 'base' for meta. The meta level handles context management, CPT, user-defined tasks and proactive service composition modules. The meta' level contains the Ss. The base level handles applications, Ss browser and the user-defined task creation environment.

Meta level also contains the 'system services' block. These are a set of services required for the working of the reflective interfaces. Discovery service is responsible for the discovery of Sss in physical or virtual proximity. Monitoring service is a service that is responsible for allocating a monitoring process to compounds and solutions where necessary. For example, in Figure 5, a highlighted version of an application and a container can be identified. It means that the application has created a container, which can be uniquely identified against that application. Containers are equivalent to the concept of 'named solutions' discussed in previous chapter. A container can have multiple solutions, and it serves as the membrane, wrapping the solutions within. The application always communicates with the meta interface via its container. The application can create reactions, using bonds and CEs from the CPT and pass them onto the 'container' as 'solutions', the 'container'

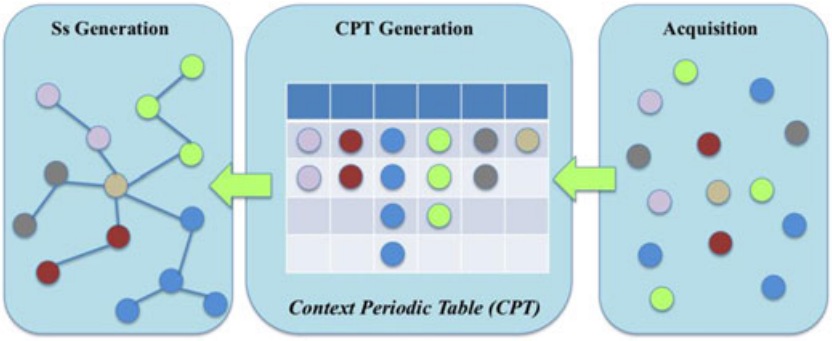

Figure 4. Context generation phases.

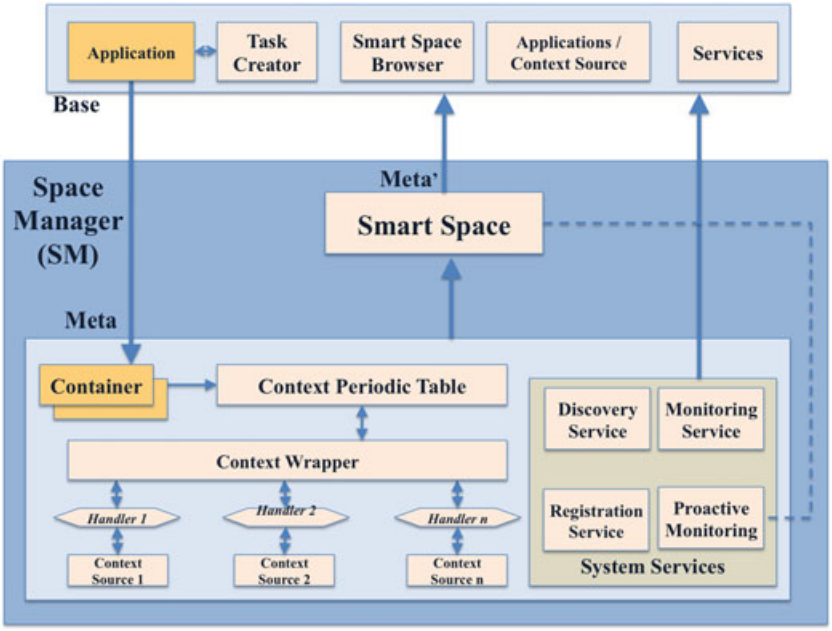

Figure 5. User plane middleware encapsulated in smart space interaction manager.

stores the reactions, and the 'monitoring service' provides it with a monitor thread that monitors the reactions and CPT.

Any change in CPT that enables a reaction for execution triggers a proactive update process towards the application, and any access from application to the middleware services and resources is also monitored, for example, if the application has floated, a reaction in the container that specifies not to use 3G in the presence of Wifi, via an abstraction bond then if the reaction is executable the container executes the reaction, and as a result enables Wifi to be more reliable if available. Figure 6 presents a summary of the reflection process that enables reactions and generates Ss graphs in terms of base and meta levels.

The next component within the SSIM is an Ss meta' interface. Ss meta' interface is a base abstraction of the CPT meta interface and offers a graph-based PSs. Therefore, it can be considered a base level to the meta interface and as a meta interface to the original base layer. The abstraction adds detail to the system and highlights multiple possibilities of reflection in terms of chemical reactions and graph-based compositions. Ss meta interface is responsible for two main interaction modes: community detection and visual exploration of the Ss. Figure 7 presents an overview of the reflective process in terms of data structure exchanged via the meta interfaces.

Base level sends in context compounds, solutions or reactions to the meta interface. The meta interface stores this input against a container with an association to an application or the Ss.

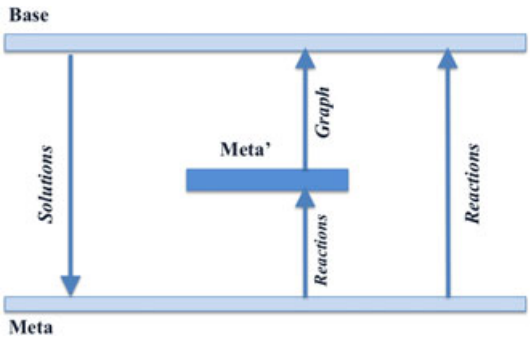

Figure 6. Space manager reflection process.

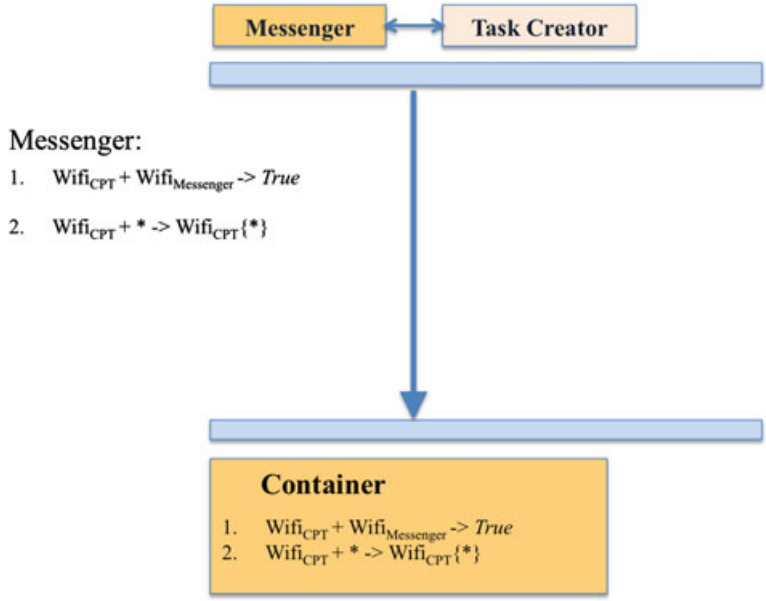

Figure 7. Example: Application using chemical model via reflection.

A user plane encompasses three forms of adaptation via the SSIM. Firstly, system level services can organise and adapt to changes in the CPT (for example, if Wifi has higher reliability than 3G and is currently not connected, a system can revert to Wifi once a network becomes available). Secondly, applications can subscribe to context events by sending CE and compounds to the 'container' and once the elements become available, middleware can perform a reaction. Lastly, the 'Ss' meta level can trigger proactive reactions without any subscription. The reactions and subscriptions are sent using the C_2A notation as discussed earlier.

Any representation mechanism can be used to communicate solutions and reactions amongst base and meta levels, as long as the scheme meets the structure defined in C_2A model. However, in this work, the developed prototype uses XML and GraphML. Both are used as representation languages to communicate CEs, CPT and Ss graphs.

To conclude, an example of the reflective process is given to show how the reflective component would work in a given scenario.

Consider a chat application titled 'Messenger'; the user of this application decides that he/she only wants to use this application over Wifi network, in case both Wifi and 3G are present. However, the developers of the application have offered no such option in the application settings. The user uses the 'Task Creator' interface to create a 'container' for the 'Messenger' application. Next, the user uses a visual representation of the CPT within the 'Task Creator' interface to create a Boolean reaction between the CPT Wifi-CE and a named variable Wifi-CE with the status variable set to 'true' and 'reliability' set to 1 (maximum). The user then creates an abstraction bond between the named Wifi-CE and any other CE of CPT's 'How' column such that Wifi-CE abstracts all other interface elements. This way, whenever the application attempts to communicate, the middleware checks for the solutions in its 'container' and only allows the application access to communication if the reactions are in the executable state.

5.1.4. Smart space interaction manager. The middleware collectively is a combination of reflective and tuple space models. The SSIM component in the 'Middleware Plane' is a tuple space-based component. It serves as a central repository for collection of all individual Ss graphs. Middleware plane consists of two main components: the SSIM, which is primarily responsible for enabling community Sss, and central storage. Storage is seen as a 'black box' in this work, and it is assumed to be capable of storing Ss graphs for all users associated with it. The SSIM consists of five sub-components. 'Space discovery' (SD) is the first in action and notifies the middleware of the existence of a personal or community SS.

In the case of PSs, SD serves as a proxy and only redirects the notification to storage. In case of a SSs, it gets the information from Ss cache for proximity triggered SSs or from Service Plane for service triggered SSs or from Storage for inferred SSs on the basis of any contextual dimension.

Next, the SD passes on relevant SS IDs to the 'space modeller' (SM). SM prunes irrelevant contextual information off the SSs and formats it in a C_2A graph. The resulting graph represents the active SSs, and a snapshot of the graph at a given instance is the 'Situation' of the SSs, where a situation has at least one CE in at least one and at maximum six columns of the periodic table. In case of an incomplete 'Situation', SM also fulfils any missing or incomplete dimension of 'Situation' to the SSs. For example, if the SSs does not have a common goal or service then SM communicates with the Service Plane to find a suitable service for the SSs. 'Space scripter' then takes over from SM and translates the SSs model to an XML script and validates the SSs. 'Space scripter' then sets up necessary resources for the execution of the SSs situation. SE then passes the execution id to the space monitor (SM), which monitors the individual entities of the SSs over the course of its lifecycle. If SSs changes, SM notifies the SM for a potential need of remodelling.

5.1.5. Service plane. The service plane contains directory and the types of services. In terms of location, service directories are maintained at access gateways, and a central repository is maintained at service plane. The central repository contains all the information about the sub-directories at gateways. The purpose of placing a service cache at access gateways is based on the fact that location serves as the main enabler for context-aware services and often services will be associated with a location therefore having a local cache can increase efficiency and lookup costs.

In terms of service types, services are classified into three categories, atomic services, 2+ services and service templates. Atomic services are services that are considered as a single component not capable of further disintegration. 2+ are services that are a product of chaining two or more atomic services. Template services are service that are complete in terms of their functional logic and are made live by inducing trigger or execution data to them (for example, a treasure hunt pervasive gaming template that needs user data and treasure data to create and execute a game for a group of users). Templates can be atomic or 2+. A service composition engine in this phase is a component that attempts a composition of services to match the needs of a potential situation identified by a Ss, in case the search within existing services returns empty.

As discussed in the previous section, the output from 'space discovery' serves as the input to 'space modeller', which has the responsibility to complete a Ss representation, especially in terms of the goal of the Ss. The goal can be a service discovery or service composition for the given space. Figure 8 shows the processes involved in orchestrating a service for a Ss, once a Ss graph (personal or social) exists, it serves as an input to the 'service discovery' mechanism, service discovery in itself is a four-step process, it first uses the Ss signature to find a suitable match amongst existing services, failing that it attempts to chain existing services, if that does not work either, it requests 'space modeller' to remodel the Ss signature, if that fails in match making, it is assumed that there is no valid service available for the Ss.

This completes the discussion related to the reflective middleware for chemistry for context awareness. This paper proposes that C_2A can be used to model and analyse IoT and the reflective middleware is the framework to realise it. In the following section, we present performance-related results of the reflective middleware.

6. IMPLEMENTATION AND RESULTS

In this section, we discuss a prototype implementation, S3, and results related to the performance of the chemical reflective middleware. S3 is a Java-based analysis and simulation tool for C_2A based on JUNG [24] graph library. S3 is a simulation tool that performs simulation-based experiments for two types of experiments, proactive and reactive. A 'context source' in these experiments is a source of information belonging to a category of context table, for example, a GPS CE belongs to the 'where' category, a calendar CE belongs to both 'when' and 'where' categories with input/output variations, an audio/video file is a context source of 'what' category. In reactive experiments variations of context

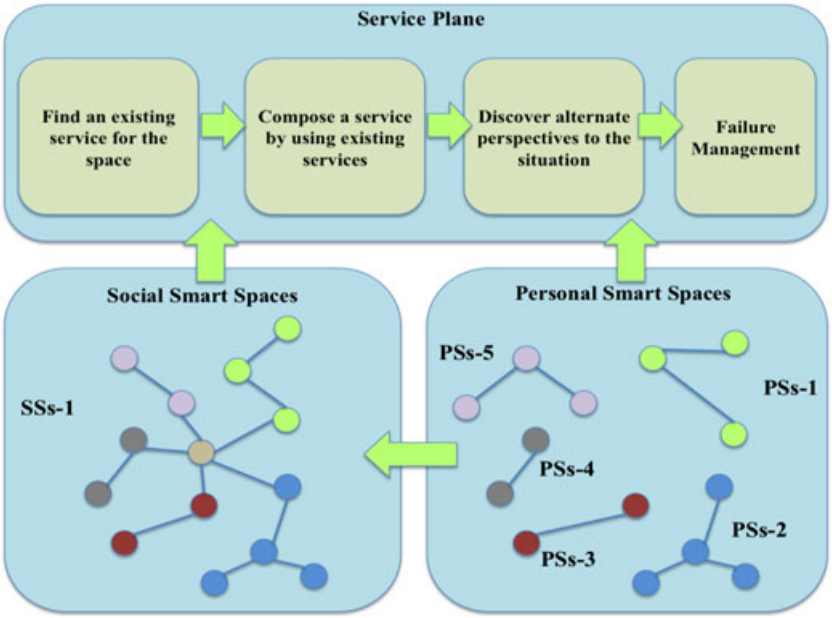

Figure 8. Service selection process (PSs-personal smart space, SSs-social smart space).

sources, occurrences and distributions are generated, and their interaction and behaviour is observed using metrics such as number of reactions occurred, distribution of reaction types, stages of a situation evolution, triggers and services reached and more. In proactive experiments, a limited number of context sources are defined along with the changes in their values and states as a function of time. The context sources are designed so as to reach a pre-defined situation following various stages along a storyline. The storyline is created by dragging and dropping CEs on the storyline canvas (Figure 9) and defining its states at various times during the story, the experiments are then run to observe if the contextual assertions are reached. In terms of S3, a context source is a Java thread with pre-modelled or random changes in its state (available/ unavailable) and values (inputs/ outputs) as a function of time and/or other reactions in the experiment. All CEs (Java thread) are placed in a container (a Java execution and scheduling service), and their interactions are observed. In this paper, we present performance-related metrics and storyline-based application scenarios (service orchestration, social network identification) orientated results at length.

In the first test, uniformly distributed increasing number of CEs were provided to S3 (simulation tool) and, the time for the creation of CPT was observed. It is worth mentioning here that although context sources used for the experiments were simulated, the creation process of creating CPT and Ss was real. Figure 10 shows the outcome; it was found that the time consumption uniformly increased with increasing CEs. However, for the largest set of CEs (120) time consumption was less than half a second. Also, 120 is a large set of CEs, and the maximum expected range of CEs, considering the high-end mobile devices with maximum sensors on board, is 10–20 CEs.

Using the same experimental setup and data in the next experiment, CPTs from previous experiments was used to generate PSs (Figure 11). It was observed that the increase in time for PSs generation with increasing CPT size was exponential as compared with a linear trend found in CPT generation. It was understandable as the process for PSs generation involves sorting and nested iterations that affect the generation time.

Different approaches could be investigated to form Ss graphs that may improve the exponential time consumption, but it is not considered part of this research, and using the current algorithm, it can be concluded that the process will take more than 1 s for approximately 130 CEs and hence may become not usable or acceptable to the pervasive scenarios. Having said that, users and devices do not have to generate Ss every time they add/remove/update context but only for the first time, and from then onwards for all changes, only the necessary parts of the table and graph are updated. Therefore, in the following experiment, see Figure 12, PSs graphs from previous experiments were

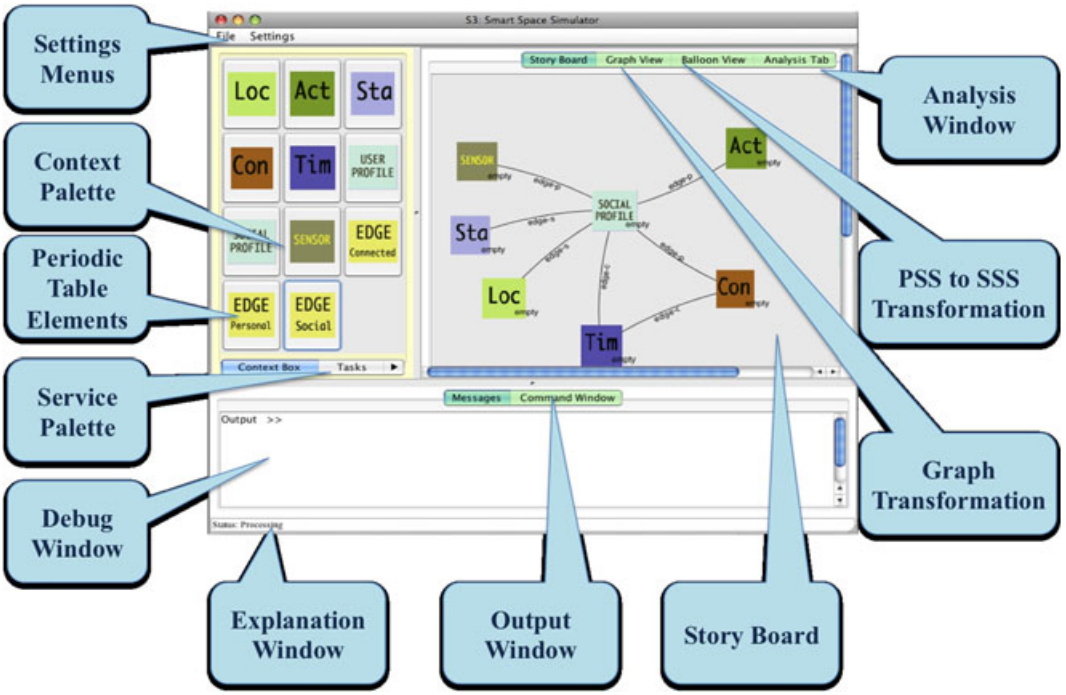

Figure 9. Screenshot of S3 simulator.

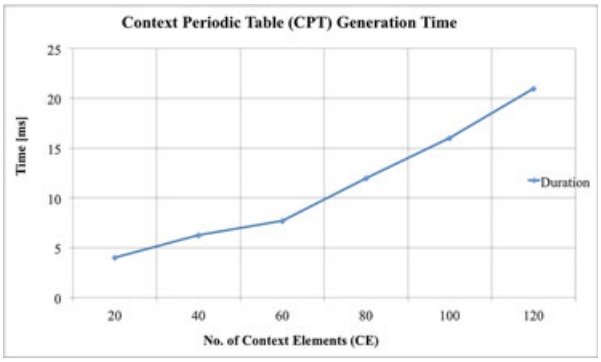

Figure 10. Periodic table generation for increasing context elements.

updated such that in the first run, one CE (same for all PSs) was added to the CPT; in the second run, two CEs were added, and in the last run, four CEs were added.

As it can be seen from Figure 12, there was a visible ripple effect on the time consumption; however, it was linear. So far, it is observed that the C_2A model performs well in context of temporal efficiency, and most of the changes in consumption are linear, which is consistent with growing input to any computational model.

In the following experiment, SSs update performance is considered for 32 PSs. The configuration is such that, for example, for the 32 PSs-based SSs, n random CEs were updated (status changed, interface changed or value changed).

Figure 13 illustrates the results of this experiment. In the first phase, for every updated CE, all of the SSs nodes and edges were reviewed; this resulted in a linear increasing time; however, in the second run of the same experiment, propagating ripple effect strategy is used to update a CE and then update all its edges as long as the comparison results in any change. This resulted in a much lower gradient linear time consumption graph.

Colour online, B&W in print

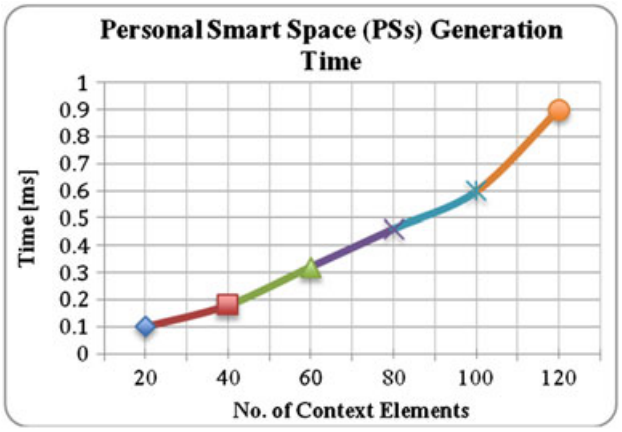

Figure 11. Smart space generation for increasing context elements.

Colour online, B&W in print

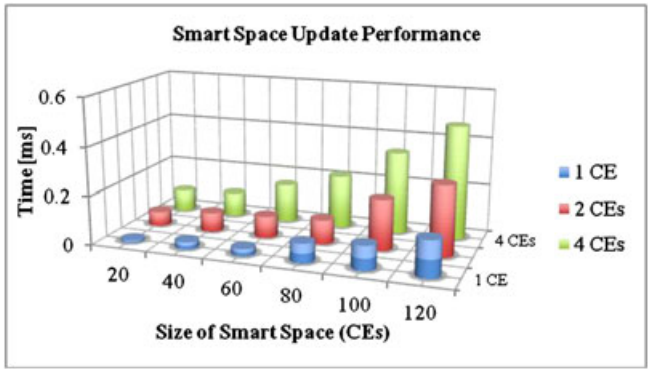

Figure 12. Smart space update performance.

Colour online, B&W in print

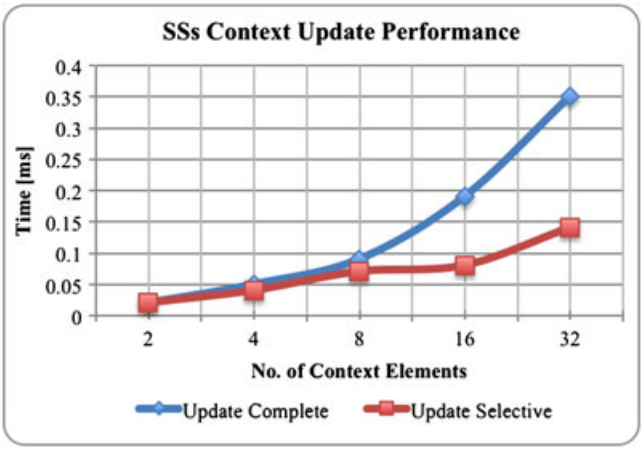

Figure 13. Social smart space update performance.

Figure 14 presents the results for service composition time consumption for the same service request parameters but increasing number of potential services to search from. It was observed that the resulting time was dependent on whether hierarchical structure found a match in the best-case

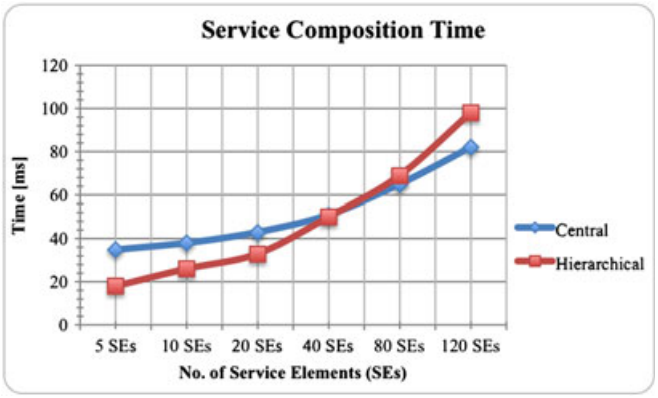

Figure 14. Service composition time consumption.

scenario (first level) or worst-case scenario (n^{th} level). Therefore the values in the graph show the mean for a set of five best-case and five worst-case runs.

We now move to qualitative experiments, and in the following experiments, we test the visualisation and modelling aspects of the chemical model using scenarios/storylines. One scenario each for proactive and reactive situations is defined with varying complexity; an example of which is shown in Table I.

The scenarios, mentioned in Table I, are used as unit tests for controlled testing and provide early inroads to the testing phase. Once the controlled testing is complete, the same battery of tests could be run for random number of users, context and services. The following section introduces a smart check-in scenario, which will be used to test if a pre-defined scenario and input result in an expected Ss or if not to what extent it fulfils the requirements of the scenario in terms of modelling context and Ss.

Smart check-in scenario

Introduction: A user titled 'User-F' enters a hotel titled 'Hotel' to book a room. User-F has pre-defined hotel preferences CE stored in his PSs, and User-F's PSs interacts with Hotel's PSs to create an SSs that enables spontaneous and personalised orchestration of room booking service.

Setup: User-F creates his personal preferences in his social profile in terms of 'location' of the hotel and features of the room in terms of, for example, 'view', 'floor', 'mini bar' and 'colours'.

Objective: Enable interaction of user-defined preferences and hotel room booking service autonomously and spontaneously to select a personalised list of rooms for the user.

Figure 15 shows visual snapshots of the PSs of both user and hotel, which are created using the simulator. User PSs has five CEs, and Hotel PSs has four CEs, where, the 'Room Preferences' CE and 'Booking' CE are of vital importance in this scenario. A user creates this CE using the canvas component of the simulator and adds it to the CPT. There are many ways of creating 'Room Preferences' CE; in this case, 'Room Preferences' CE takes as input location, and if the location is the same as that of 'Hotel' (assumed to be known), then it turns its status to 'On' and outputs 'Preferences' (in this case, a comma-separated string containing key-value pairs but can be anything

Table I. Storyline-based case study scenarios.

Reactive-personal	Proactive-social
Scenario 1: smart check-in scenario A user's PSs interacts with the PSs of a hotel to select room for his pre-stored preferences	**Scenario 2: airport arrival scenario** PSs of two users with different language preferences interacts with airport's PSs to find a taxi service

PSs, personal smart space.

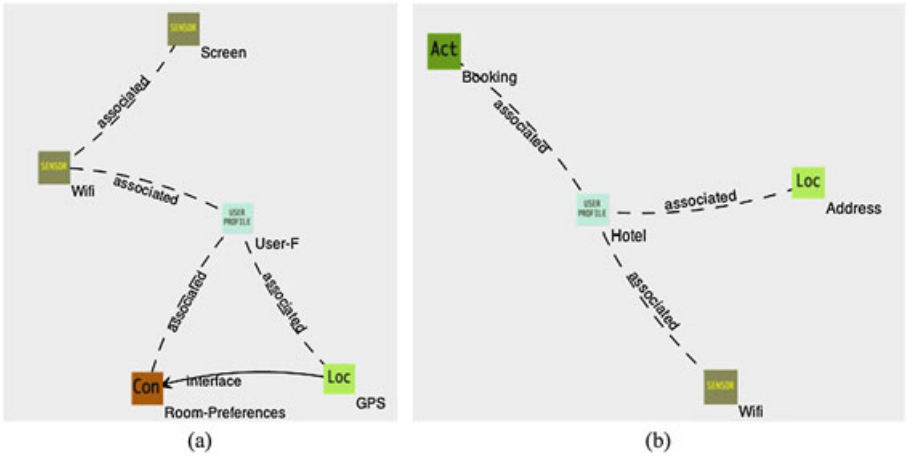

Figure 15. (a) Personal smart space of User titled 'F' (b) Personal smart space of hotel titled 'Hotel'.

depending on the implementation). For the 'Hotel' space, 'Booking' CE is responsible for making hotel bookings and takes as input 'Preferences'. Booking CE outputs a displayable list. Both the PSs on every change in CE update a thread in the simulator, which in turn updates the central repository. The location of User-F is then manually updated to that of the 'Hotel', in this case, dummy coordinates '(1.234, 2.345)'. At this point, the two PSs become eligible for reaction and generation of SSs.

Figure 16 shows the SSs graph of 'User-F' and 'Hotel'. In the process of generating SSs, two major proximity bonds are formed, highlighted in Figure 16. Firstly, the 'Room Preferences' CE of user space and the 'Booking' CE of the hotel space react on the basis of the same interfaces and form a 'proximity-interface' bond. Secondly, 'Booking' CE reacts with the 'Screen' CE using the interface bonds, and the two reactions enable selection of a personalised list of rooms being displayed to the User-F. The bond between the 'Address' CE of 'Hotel' and 'Room Preferences' of User-F is worth noticing as it enables the first reaction of noticing that both the 'WHERE' inputs to the CE are same; hence, the reaction is possible.

Furthermore, apart from explicit reactions highlighted in the SSs, there can be implicit reactions that can be derived from the SSs graph, for example, as both the PSs have an active Wifi-CE, when the

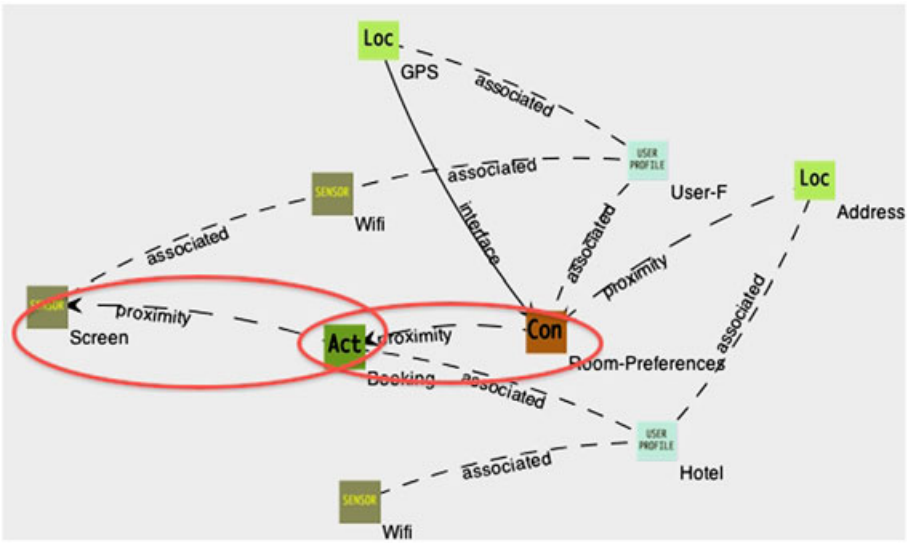

Figure 16. Scenario 1 – 'User-F' and 'hotel' social smart space.

'Booking' CE outputs the list of rooms to User-F, its PSs is responsible for finding an access medium to receive the content, for which, it checks its 'How/Sensor' elements searches for a CE that can send and receive a 'connection'.

Airport arrival scenario

Introduction: Two users titled 'User-F' and 'User-G' arrive at an airport titled 'Airport'. Both users have language preferences defined in their PSs as 'French' and 'German'. Airport has two SEs 'Translation-Service' and 'Weather-Service'. 'Translation-Service' depends on language and content to work and 'Weather-Service' needs location for execution. Both the services output displayable content. Moreover, another user titled 'User-Pickup' is at airport to pick 'User-G' and has a corresponding user-defined activity in his PSs.

Setup: When the tuple space detects a community of these four PSs, it enables the SSs and formulates possible contextual reactions in the SSs. 'Weather-Service' signature is (Input: location, Output: content). 'Translation-Service' signature is (Input: Language, Content Output: Content).

Objective: In this scenario, the experimental objective is to observe whether users are able to access personalised translation services. Moreover, is it possible for the 'Weather-Service' to spontaneously and autonomously deliver personalised weather content? Lastly, can the SSs help 'User-G' and 'User-Pickup' to discover and communicate spontaneously?

Figure 17 presents a snapshot of the PSss involved in the scenario. The CPTs of the four participants are kept to the minimum necessary CEs to restrict the size of the SSs in terms of nodes and edges. All

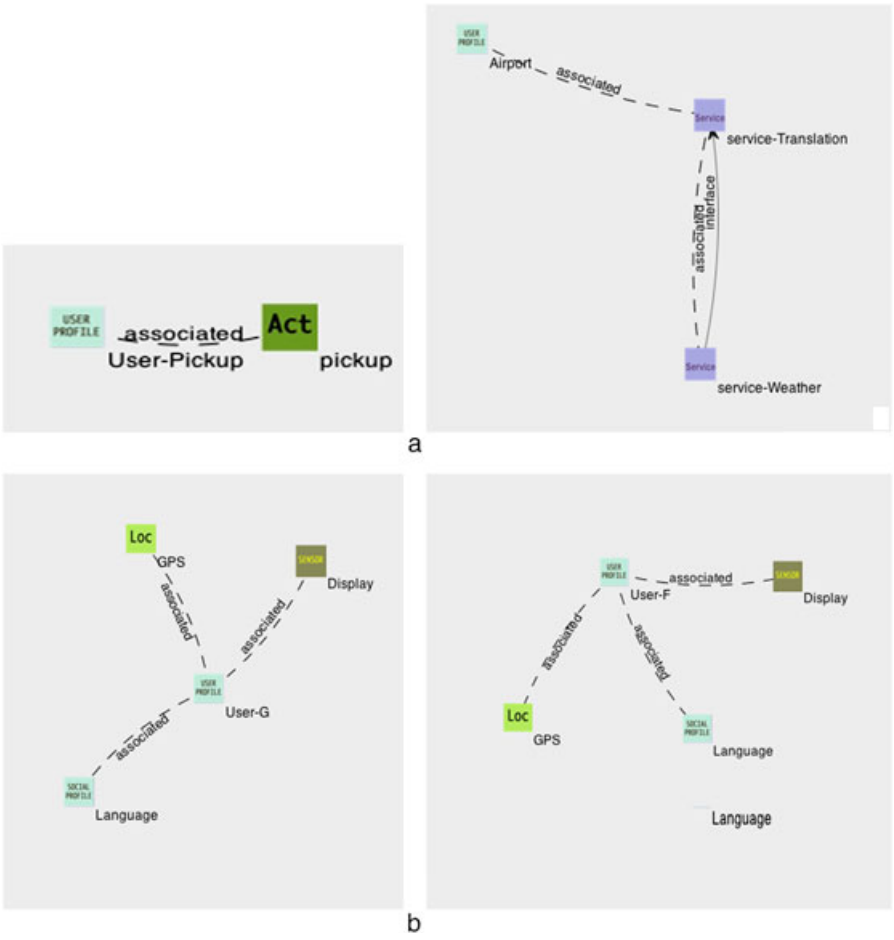

Figure 17. Snapshots of personal smart spaces of User-F, User-G, Pickup and Airport.

four PSs are mainly consisting of ‘association’ bonds with exception of Airport PSs whose ‘Weather-Service’ SE has an interface bond with the ‘Translation-Service’ as the ‘Weather-Service’ outputs ‘content’, which is also a possible input to the ‘Translation-Service’. The bonds formed here are only on the primitive basis of interface type matching; semantics of matching procedure is not part of the current research and is assumed sufficient.

Figure 18 shows the snapshot of the SSs when the tuple space detects a location-based community and forms a SSs. The process identifies any possible bond formation between the CEs of newly formed community, and as it can be seen, a variety of ‘proximity’ bonds are added to the SSs. Proximity bonds can be of any type such as ‘interface’ or ‘abstraction’; however, they are labelled as proximity as they are enabled because of the proximity of PSss.

As an early observation, it can be seen that for four entities and a collection of 13 CEs, the resulting SSs has become much more complex, and the edges have become difficult to trace. This issue is discussed at length in the usability section, presented later. For the sake of this experiment, bond-based filtering is applied to the SSs, and the criteria for filtering are taken as ‘proximity’ bonds. Figure 19 presents the filtered visualisation of the SSs.

There are various interesting reactions to be discussed in Figure 19. At the bottom of the snapshot, a proximity bond between ‘User-G’ and ‘pickup’ activity is formed; the outcome of the pickup activity is a ‘Boolean’ value; therefore, as soon as the SSs forms a proximity bond between ‘pickup’ and ‘User-G’, ‘User-pickup’ can use the SSs visual component via the ‘association’ bonds to access the ‘GPS’ CE of ‘User-G’ to check the exact location.

Next, it can be noticed that both ‘Translation’ and ‘Weather’ service have created a proximity-interface bond with any display that they could find in the SSs as they both output displayable content. However, in the given scenario, they both contain ‘weather’ data in three different languages depending on the path taken in the graph. In context of autonomous reactions, this presents a conflict, as the display has to decide which content to display. In C_2A, such conflicts in autonomous mode are handled by the priority of the CEs. For example, in Figure 17 and Airport PSs, it can be seen that ‘Translation’ is given higher priority than ‘Weather’ as it is closer to the

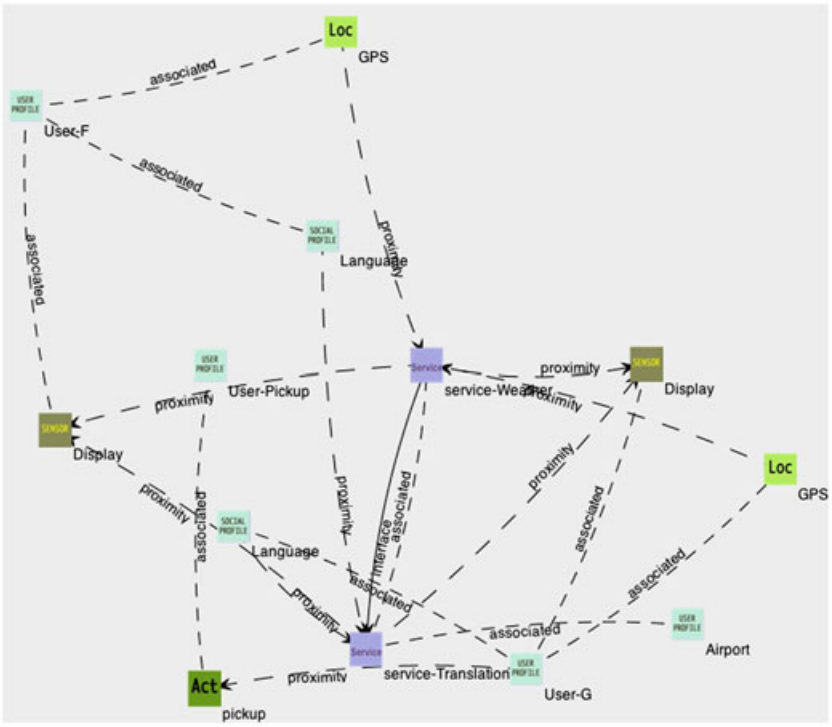

Figure 18. Social smart space of the ‘Airport scenario’.

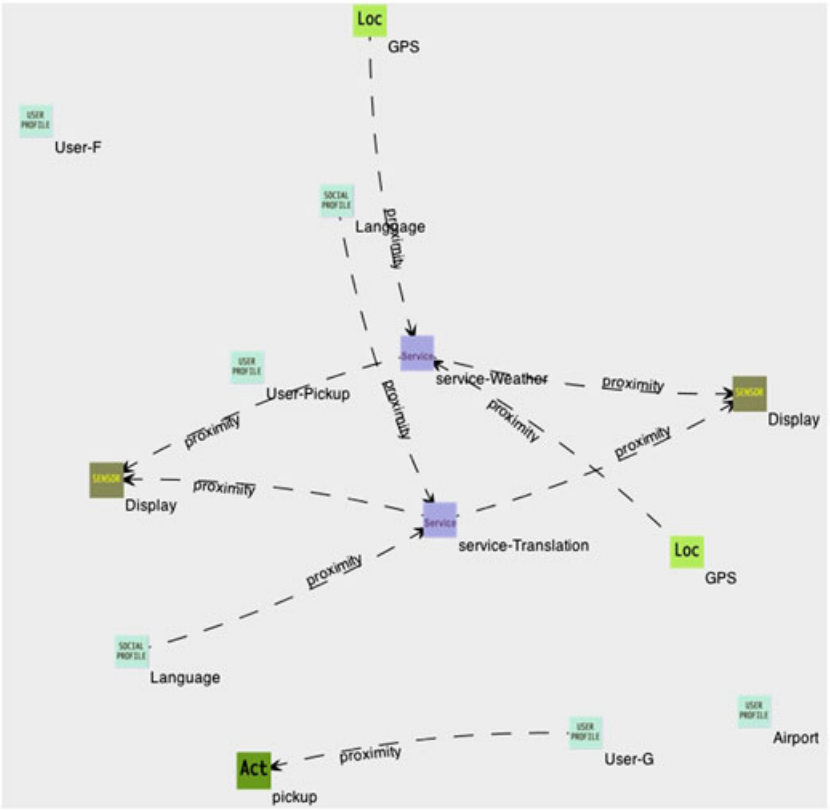

Figure 19. Social smart space of the 'Airport scenario' filtered by proximity bonds.

root profile element. Therefore, the orchestration path involving 'Translation' service is given the priority, which means that User-G's device will display the weather in German and User-F's display will display the weather in French as the social community exists in parallel enabling a rendezvous point for User-G and User-Pickup.

This experiment has contributed towards analysing creation and existence of personal and SSss in parallel. However, it also highlights the exponentially increasing visualisation complexity of the visualisation aspect. Moreover, it also highlights conflict handling via the reliability metric of the CPT.

As there is not considerable related work that addresses context awareness using chemical computation. The evaluation phase in this research aimed at answering three main questions. (i) Does the chemical model yield any useful results? (ii) Is the chemical model capable of handling dynamic, growing and large-scale context and information sources, such as IoT? And (iii) can the chemical model be formally verified? These questions were discussed, and the results were encouraging as the storyline-based assertive scenarios suggested reactively modelling and proactively orchestrating dynamic and continuously evolving situations such as context awareness, and IoT could be handled. The results in this paper aimed at answering question 2. Most of the experiments resulted in a non-linear efficiency, which suggests that growing number of information sources will make the reasoning and modelling efficiency challenging for huge sets of data. At the same time using the atomic nature of chemical reactions, pluggable reaction types and the possibility of enabling parallel reactions independent of each other, distributed and concurrent algorithms can be investigated to form multi-tier containers to spread out reactions for Sss as large as IoTs.

The results show that the reflective model is capable of realising chemical style interactions amongst entities that can be users, devices or information. The results also show that the reflective middleware is sufficiently efficient to keep the reactions and analysis close to being real time, both in generation and update scenarios.

7. CONCLUSIONS AND FUTURE DIRECTIONS

On the basis of the results discussed in this paper, it can be summarised that nature-inspired chemical models hold the potential to model interaction, visualisation and analysis in next generation pervasive networks and IoTs. Furthermore, reflective middleware models can efficiently realise the chemical computation model. Together, they hold great potential in addressing challenges associated with future computing and communication systems, although research into this branch of nature-inspired computing models is somewhat restricted at present. To address this shortcoming, we have proposed a chemical computing model for context-based pervasive environments and evaluated it for efficiency and scalability. At the same time, this paper also highlights the fact that there are not many higher-level languages and middleware tools for chemical computing. We simulated and realised the chemical computing model using reflection principles, where reflection-enabled reactions and adaptation, and tuple space enables solutions and containers. It was observed that the linear timelines can be inefficient and time consuming in complex scenarios, suggesting that an alternative can be to explore peer-to-peer architectures to realise the collection, generation and propagation of SSss.

REFERENCES

1. Ashton K. That 'Internet of things' Thing. *RFID Journal* 22 July, 2009.
2. Privat G. Extending the Internet of things. *Digiworld Economic Journal* 2012; **87**:101–119.
3. Aggarwal CC, Ashish N, Sheth A. The Internet of things : a survey from the data-centric. Managing and Mining Sensor Data, (IBM TJ WRC) Aggarwal CC (ed.). Springer: US, 2013; 383–428.
4. Gershenfeld N, Krikorian R, Cohen D. The Internet of things. *Scientific American* 2011; **291**(4):76–81.
5. Zakriti A. Service entities model for the Internet of things. *International Conference on Multimedia Computing and Systems (ICMCS)*, 2011; 1–5.
6. Banâtre JP, Fradet P, Radenac Y. Principles of chemical programming. *Electronic Notes in Theoretical Computer Science* 2005; **124**(1):133–147.
7. Dittrich P, Ziegler J, Banzhaf W. Artificial chemistries–a review. *Artificial life* 2001; **7**(3):225–75.
8. Nemeth Z, Perez C, Priol T. Workflow enactment based on a chemical metaphor. *Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05)*, 2005; 127–136.
9. Németh Z, Pérez C, Priol T. Distributed workflow coordination: molecules and reactions. *Workshop on Nature Inspired Distributed Computing. Proceedings of 20th IEEE International Parallel & Distributed Processing Symposium*, 2006.
10. Wang C, Pazat JL. Using chemical metaphor to express workflow and service orchestration. 2010; 1504–1511.
11. Di Napoli C, Giordano M, Németh Z, Tonellotto N. Using chemical reactions to model service composition. *Proceeding of the second international workshop on Self-organizing architectures*, ACM: New York, New York, USA, 2010; 43–50.
12. Banâtre JP, Fradet P, Radenac Y. Programming self-organizing systems with the higher-order chemical language. *International Journal of Unconventional Computing* 2007; **3**:161.
13. Tschudin C, Yamamoto L. A metabolic approach to protocol resilience. *Proceedings of the 1st IFIP Workshop on Autonomic Communication (WAC'04)*, Berlin, Germany, 2004.
14. Lin H, Kemp J, Molina W. Parallel computing in chemical reaction metaphor with tuple space. *Journal of Computer Science and Security* 2010; **4**:149–159.
15. Capra L. Reflective Mobile Middleware for Context-Aware Applications. University College London, 2003.
16. Kuo Z, Wuyanni W. PURPLE: a reflective middleware for pervasive computing. *Third International Conference on Information Technology and Applications (ICITA'05)*, 2005; 64–69.
17. Grace P, Blair GS, Samuel S. ReMMoC : a reflective middleware to support mobile client interoperability. *International Symposium of Distributed Objects and Applications*, 2003; 1–18.
18. Chan ATS, Chuang SN. MobiPADS: a reflective middleware for context-aware mobile computing. *IEEE Transactions on Software Engineering* 2003; **29**(12):1072–1085.
19. Ramachandra K, Ali HH. Hydra: a new approach for integrating various wireless environments. *Computers and Communications, 2005. ISCC 2005. Proceedings. 10th IEEE Symposium on*, 27–30 June 2005; 961–968. DOI: 10.1109/ISCC.2005.81
20. Dobre C. CAPIM: a platform for context-aware computing. *2011 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, 26–28 October 2011; 266–272.
21. Riboni D, Bettini C. COSAR: hybrid reasoning for context-aware activity recognition. *Personal Ubiquitous Computing* 2011; **15**(3):271–289.
22. Gaddah A, Kunz T. A survey of middleware paradigms for mobile computing. *Technical Report SCE-03-16*, Carleton University Systems and Computing Engineering, 2003.
23. Hong J, Suh E, Kim S. Context-aware systems: a literature review and classification. *Expert Systems with Applications* 2009; **36**(4):8509–8522.

24. Java universal network/graph framework (JUNG). http://jung.sourceforge.net/
25. Kon F, Costa F, Blair G, Campbell RH. The case for REFLECTIVE middleware. *Communications of the ACM* 2002; **45**(6):33–38.
26. Qilin L, Wei Z, Mintian Z. Research on reflective middleware system. *2009 International Forum on Information Technology and Applications*, May 2009; 3–6.
27. Capra L, Blair GS, Mascolo C, Emmerich W, Grace P. Exploiting reflection in mobile computing middleware. *ACM SIGMOBILE Mobile Computing and Communications Review* 2002; **6**(4):34–44.
28. Mokhtar SB, McNamara L, Capra L. A middleware service for pervasive social networking. *Proceedings of the International Workshop on Middleware for Pervasive Mobile and Embedded Computing*, 2009; 1–6.
29. Robinson R, Henricksen K, Indulska J. XCML: a runtime representation for the context modelling language. *Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops, 2007. PerCom Workshops '07*, 2007; 20–26.