Decoupling Cognitive Agents and Virtual Environments

Jeehang Lee, Vincent Baines, and Julian Padget

Department of Computer Science, University of Bath,
Bath, BA2 7AY, United Kingdom
{j.lee,v.f.baines,j.a.padget}@bath.ac.uk

Abstract. The development of and accessibility to rich virtual environments, both for recreation and training activities leads to the use of intelligent agents to control avatars (and other entities) in these environments. There is a fundamental tension in such systems between tight integration, for performance and low coupling, for generality, flexibility and extensibility. This paper addresses the engineering issues in connecting agent platforms and other software entities with virtual environments, driven by the following informal requirements: (i) accessibility: we would like (easily) to be able to connect any (legacy) software component with the virtual environment (ii) performance: we want the benefits of decoupling, but not at a high price in performance (iii) distribution: we would like to be able to locate functionality where needed, when necessary, but also be location agnostic otherwise (iv) scalability: we would like to support large-scale and geographically dispersed virtual environments. We start from the position that the basic currency unit of such systems can be events. We describe the Bath Sensor Framework, which is a middleware that attempts to satisfy the above goals and to provide a low-latency linking mechanism between event producers and event consumers, while minimising the effect of coupling of components. We illustrate the framework in two complementary case studies using the Jason agent platform, Second Life and AGAVE (a 3D VE for vehicles). Through these examples, we are able to carry out a preliminary evaluation of the approach against the factors above, against alternative systems and demonstrate effective distributed execution.

1 Introduction

Programming the environment in which a multiagent system is situated has been and continues to be an active research issue [36]. From this perspective, many rich open systems, formed from networked 3D virtual environments such as online games, non-gaming applications or some entertainment context are all potential (programmable) environments, since they offer sufficient variety to simulate real and semi-real world situations. Second Life [27] is an obvious representative example of a 3D virtual environment: it provides a sophisticated, dynamic and realistic virtual world as if duplicating the modern human society with avatars and 3D objects [26]. Such a virtual world may encourage advances in agent intelligence, through the demands of sensing and interaction, as agents are situated in more and more complex, dynamic or realistic environments.

However, the integration of agent software and rich environments creates a range of challenges, arising not least from the variety of each and that neither is typically designed to interact with the other. For example, the purpose of the Second Life is to provide an avatar in a networked 3D virtual environment, that it is expected a human will control, so it does not explicitly take into account either the use of AI or integration with other applications. The Jason agent platform [10] is similarly placed: its objective is to provide a BDI-based a deliberative reasoning engine for agent research, so it does not consider standard programming interfaces for other environments.

As a result, the research on agent-environment programming has mostly relied on tightly coupled approaches, characterised by using a specific ontology, protocol and interface that are particular to one system [1, 7–9, 22, 31, 41, 42]. Such a lack of interoperability is possibly not beneficial overall for the development of agent intelligence because there is little scope for the agent that is built for one VE to be exercised in another and so the agent is unavoidably mono-cultural. Besides, such tightly connections between agents and particular environments must somehow inhibit further potential applications arising from alternative agent-environment combinations. We believe that a way forward from this situation may be possible through an appropriate form of middleware.

Thus, our objective is to describe and to demonstrate a kind of integration middleware that serves not only to loosen the coupling between agents and environments, but also to make it possible to consider the connection of any kind of agent and any kind of environment. In software engineering pattern terms, we outline a façade for each agent platform and each environment, where each communicates with the other by means of events (in effect, asynchronous message passing), facilitated by the use of a publish-subscribe server. This constitutes the essence of the Bath Sensor Framework (BSF), which provides the means to link software components independently of programming language, platforms or operating systems, so in principle offering good accessibility, distribution and scalability as an agent-environment integration framework. Performance is a more delicate issue that will take time and experience to establish, depending on the communications overhead (the pub/sub server) and – more likely to dominate – on the decision-making cycle of the agent, although this will clearly depend on the sophistication of the agent architecture.

The remainder of this paper is organised as follows. The overall system design is be described in section 2. Section 3 presents the case studies and their results with example applications. We finish with a brief survey of related work (section 4) followed by discussion and future work (section 5).

2 System Design

In this section, we describe the whole system architecture and how it can integrate an agent platform with a virtual environment. In particular, we will

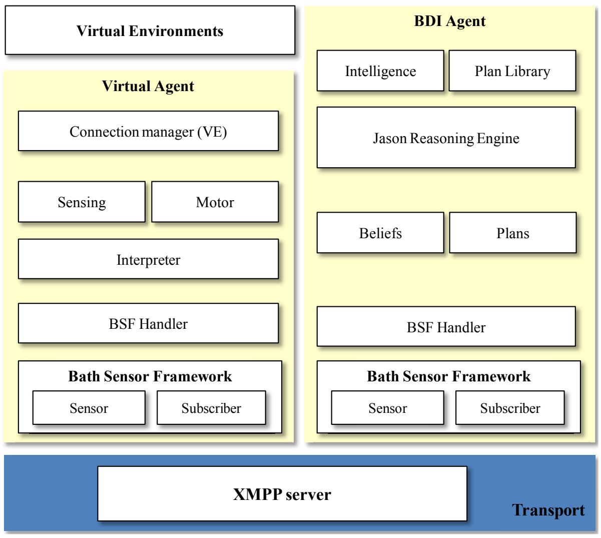

Fig. 1. Overall System Architecture

demonstrate the interaction between the software components and describe the programming model.

For our experimental set-up, the collection, distribution and exchange of data is performed by using publish/subscribe between event producers and consumers via the Extensible Messaging and Presence Protocol (XMPP), an open standard communications protocol [19, 20]. Although XMPP is often cited as a component in real-time (web) systems [40], there is little quantitative evidence to back this up. In consequence, we have carried out some preliminary evaluation (see 3.3) and we return to this issue in related work. For an agent platform, we use the Jason BDI framework and as virtual environments, we use Second Life and the AGents and Autonomous Vehicles Environment (AGAVE).

Any of these components (Jason, SL, AGAVE, XMPP) may be substituted in pursuit of a better fit with the requirements set out earlier, but the primary focus of this paper is our evaluation of the adequacy of the BSF, instantiated as outlined, as a solution to the issues identified above pertaining to agent-environment programming. In doing so, we present case studies that connect Jason agents, Second Life and AGAVE, and discuss how to accommodate differences in platform and programming language.

2.1 Overall System Design

The essence of the design is that the agent platform is decoupled from the virtual environment by means of a publish-subscribe messaging server – in this case, XMPP – as shown in Figure 1.

In the case of Second Life, the virtual agent is created using the openmetaverse library (LIBOMV [30]), which also provides the connection to the Second Life server. The role of the virtual agent is to interpret the actions received from the BDI agent, and then carry out the resulting "physical" actions. In the other direction, the virtual agent perceives the environment and the percepts are delivered to the BDI agent via BSF, where it is becomes a belief that influences the agent's reasoning process.

Clearly the BSF plays a key role in facilitating the interaction between the two components. In particular, through the imposition of a simple communication API, a java-based agent platform and a virtual agent, in this case written in a different language and running on a different platform, can interact with one another. We now explain in more detail about the sensor framework.

2.2 Bath Sensor Framework

The Bath Sensor Framework (BSF) is an abstraction layer for data collection, distribution and exchange built upon XMPP technology. The primary task for which the framework was conceived is the effective collection of data from numerous physical or logical sensors, and its subsequent distribution to the relevant devices or software connected to the XMPP server. The data itself is represented in RDF, although this is not mandatory: the XMPP message structure is just a HTTP body and can be any representation that is suitable.

XMPP is an open standard communication protocol built upon a set of open XML technologies [19, 20]. It is intended to provide not only presence and real-time communication services, but also interoperability by exchanging any type of data in cross domain environments by means of nodes in the XMPP server. To this end, it supports 1-to-1, 1-to-many, and many-to-many data transport mechanisms, so that any data may be be transferred from anywhere to anywhere [6]. Its flexibility, performance and lightweight nature have lead to XMPP being chosen to support research in a diverse range of fields, including Many Task Computing [39], bio-informatics [43] and Cloud Computing [5], as a data distribution service in preference to HTTP or SOAP services.

The above features suggest a number of advantages over pure TCP/IP connections. The latter typically require quite careful set-up and can be fragile where the connection graph is not simple. Moreover, TCP/IP is primarily for 1-to-1 connections, so every additional connection needs an independent additional socket whenever multiple software components are integrated into the main system. In contrast, XMPP provides a star- or bus-like connection model to resolve the m-to-n problem, but through the node abstraction within the server, allows the set-up of multiple virtual circuits. Furthermore, the data producer does not need to know the consumer's identity to set up the connection and through the server's mediation of the connection, the system acquires a degree of fault-tolerance, and permits the observation of system behaviour by third parties, rather than having to replicate such mechanisms in each component. Thus, we conclude that XMPP offers several attractive features, which is why we have chosen to base BSF upon it.

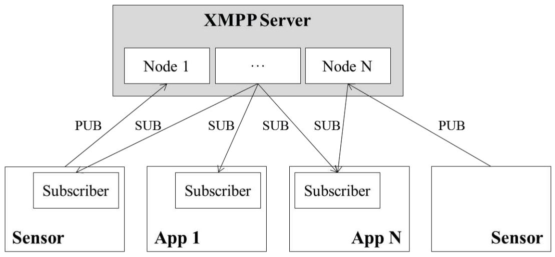

Fig. 2. The System Architecture of Bath Sensor Framework (BSF)

For our purposes, the most notable feature of XMPP and hence the BSF is its provision of a publish/subscribe mechanism. BSF supports these operations by means of *sensor* and *subscriber* classes. When the *sensor* is created in the application which has a role as a data source, a corresponding node is also created in XMPP server. Once created, the application can publish the data sensed from the real or virtual world via the node. If the *subscriber*, which is created in another application, which has a role as a data consumer, sets up a subscription request on that node and registers a corresponding handler in its application, then the data will be transferred from one application to the other. As noted earlier, the data is represented in RDF and published data can be stored in a triple-store (in our case OpenRDF) so that historical data can be retrieved on request from the *subscriber* using the SPARQL query language[1]. In this way, the BSF supports a form of messaging passing with unstructured data between multiple software components.

A particularly valuable aspect of XMPP/BSF is its relative independence from both operating systems and programming languages so that more general programming environments can be provided for users attempting to combine heterogeneous software components. Thus, regardless of language its interfaces reveal the same classes, methods, data structures and interfaces, so that all kinds of applications or libraries can be integrated relatively easily just by adding the classes inside applications.

As can be seen, the features of this framework present a simple and flexible programming environment for heterogeneous software components, with a good level of a accessibility in terms of a simplicity of protocol and ease of connection, performance, and distribution. Consequently, we show how the BSF can facilitate the integration of a cognitive architecture for virtual agents. The next section discusses the programming model of the BSF.

2.3 Programming Model of Bath Sensor Framework

The Bath Sensor Framework can equally be applied to data exchange between distributed software components as to sensor based applications. The perspective

[1] Other (structured, relational) databases may equally be connected to the Openfire server and accessed by SQL queries.

of this section is limited to the former. The analogy we draw, in making the connection between BDI agent, virtual agent and virtual environments, is that the virtual agent can be viewed as a sensor for the percepts from the environment. In this context, the *sensor* object is instantiated in a virtual agent in order to collect percepts for the BDI agent. Conversely, the BDI agent needs a *subscriber* object to receive the percepts and subsequent reasoning over acquired beliefs.

In a data consumer such as in the BDI agent, the *subscriber* object has to be instantiated inside the BDI agent. The C# version of this example is identical modulo the grammar of the programming language.

The objective of the design is that it should suffice just to put the *sensor* and *subscriber* object in a wrapper around whichever software component it is desired to integrate into the event processing framework.

3 Case Studies

The aim of the case studies is to demonstrate how the BSF enables the integration of agents and virtual environments; experiments in the actual usage of agents and VEs will be part of future work. In particular, the studies focus on the impact of the use of BSF on the integration of agent platform and virtual environment, in respect of some desiderata for computational models such as generality, modularity and dynamic extensibility [36]. A complementary aspect is the increased capacity for distribution of the components of the software architecture, so that it is not so tightly coupled and that the addition or removal of components is straightforward.

In the preceding section, we outlined how *sensor* and *subscriber* objects are incorporated into a BDI agent and a virtual agent. For the following discussions, the components are (i) Jason, providing a BDI agent, (ii) the openmetaverse library, providing a virtual agent, and (iii) Second Life and AGAVE, providing a virtual environment all linked by the BSF.

3.1 Case Study 1 : Jason Agent and Second Life

The goals of this study are two-fold: (i) to demonstrate the integration of Jason agents with avatars in Second Life via BSF (ii) to identify appropriate mechanism for the control of avatars via BSF, through exploratory scenarios. We also take into account interaction not only between avatars controlled by Jason agents, but also those between humans in real world and avatars in virtual worlds. This latter direction will be explored in more depth as part of future work.

The brief scenario for this section is as follows: one avatar controlled by a human in Second Life server says ‘hello’ to a Second Life avatar governed by Jason agent. In what follows, we refer to the Jason controlled avatar as the Second Life Bot (SLB). When the SLB receives the greeting message, the SLB sends it to the Jason agent over XMPP via the *sensor*, where it is received via the *subscriber*. The Jason agent then updates the percepts, and performs one cycle of reasoning. As a result, the belief ‘hello’ triggers the plan ‘bow’, and appropriate

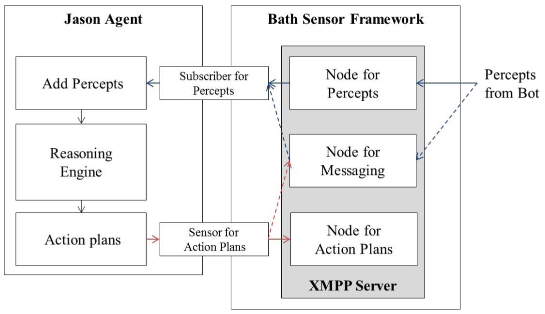

Fig. 3. Basic Operation between Jason agent and Bath Sensor Framework

actions are sent to the SLB. Finally the bot does a 'bow' animation by means of the openmetaverse library after interpreting the action plan 'bow'. Interpretation of the action plans means the conversion of actions from Jason to Second Life animation action(s). For example, if 'bow' is received from Jason, then SLB looks to see whether 'bow' is defined in the action map: if so it perform that animation. More commonly, an action plan is likely to be composed of several atomic actions (or animations) in SLB.

A notable aspect of this scenario is that two heterogeneous software components are able to interact by means of the BSF: because the openmetaverse library is in C#, so too is the SLB, but Jason is Java. Previous work has been able to integrate them by means of the .NET framework [31], but this requires all the components to be in the same location, on a specific platform and also couples them quite tightly.The C# interface to BSF is achieved by an extension of the jabber.net library [25], while the Java interface is built on the Smack library [35], although this is just one of several available Java libraries for XMPP. We are currently using the OpenFire [34] XMPP server, although again there are several other candidates.

Jason Agent and Bath Sensor Framework. There are two ways in which to use Jason agent reasoning engine. The most straightforward is to subclass the environment class, which provides interfaces that are triggered by internal events during the reasoning cycle. This is a quick method of construction, but has limitations: (i) the dynamic update of percepts is impossible because the update interface only can be triggered by the Jason reasoning engine, so external update events from the environment cannot update percepts directly (ii) the other problem is that action execution in the environment class is limited to those actions defined in the base environment class, whle those in the subclass are inaccessible. An alternative approach is to use the Jason agent reasoning engine by subclassing the AgArch class from the Jason class library. The latter is a more general technique for deploying embodied AI into rich environments. Hence we choose this latter approach.

Figure 3 sketches the basic operations between a Jason agent and the BSF. The Jason agent is extended, using the AgArch class, with the *sensor* and

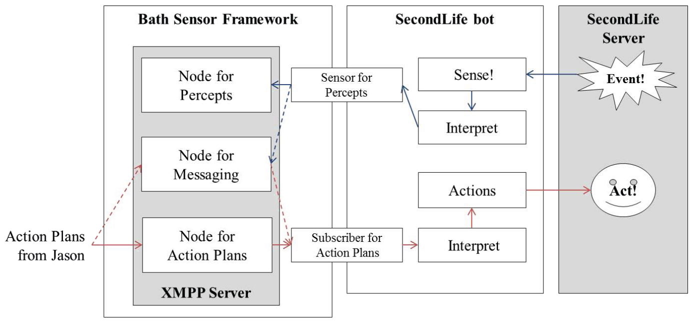

Fig. 4. Basic Operation between Second Life Bot and Bath Sensor Framework

subscriber objects. Percepts from the SLB are received by the *subscriber* object, which results in updates to the beliefs. The next reasoning cycle utilises these beliefs to retrieve an action plan and the *sensor* object publishes the plan to the SLB.

Second Life Bot and Bath Sensor Framework. The open metaverse library provides a set of APIs to program the avatar in terms of creation, appearance, movement, communication – verbal and non-verbal – and interaction with each other, in the same way as the Second Life official viewer application. Thus, with openmetaverse, it is possible to program complex compound actions in the avatar.

Once logged in, the SLB appears as an avatar in Second Life. As such, it has an identity and can move and interact with other participants, as well as perceive events taking place nearby. Consequently, all events occurring in Second Life are detectable in the openmetaverse library and delivered via a callback mechanism to the *sensor* object in the SLB, which collects them and publishes them for the Jason agent to receive. On the other side, the *subscriber* object receives (subscribes to) the action plans from the Jason agent. These are then translated into sequences of atomic actions, which are a combination of defined actions in openmetaverse or user-defined actions. As a result, the SLB carries out these actions in respect of other participants or its environment.

3.2 Case Study 2 : Jason Agents and AGAVE

This case study also has two goals, first to demonstrate the use of Jason agents controlling simulated vehicles within a virtual environment, and second, to replace the simulated vehicle with a physical robot vehicle, but all controlled via the Bath Sensor Framework and XMPP messaging. The AGAVE framework evolved from earlier work based on a Jason-based virtual tank simulation called TankCoders [21]. The solution has now been redesigned, with the Bath Sensor Framework at its core, with integration to the jMonkeyEngine simulation package (to provide a 3D scene), the AllegroGraph data store (for replay and performance analysis), and the Jason BDI framework, to form the AGents and Autonomous Vehicles Environment (AGAVE). Vehicles are controlled via XMPP messages, and report information back via the same mechanism using the BSF. The aim of this work is the construction of scenarios for the exploration of vehicle convoys [4] in the context of automatic driving for their potential benefits in fuel consumption and fatalities, as well as improving traffic efficiency [16].

AGAVE, Jason, and Virtual Vehicles. Unlike the system described in [21], the AGAVE simulation components here are decoupled from Jason for the sake of increased flexibility in where the code is run and the requirements placed on the underlying implementation. As the BSF supports a Resource Description Framework (RDF) data model, the ontology of message exchange is defined, allowing alternative components to be easily integrated. For example, vehicles are expected to publish geo-spatial updates to the BSF, and subscribe to set of defined control messages (e.g. setOrientation, setSpeed). Of course, how those operations are realized depends on the end device, allowing easy substitution for simulated by real vehicles. Similarly, components such as the 3D-view scene subscribe to vehicle updates, and display those positions via the simulated scene.

A vehicle is currently a simple simulated abstraction that publishes its spatial location at a predefined heartbeat, and updates its location based on current orientation and speed. When started, the simulated vehicle is provided with a name, and will publish its spatial position with this name included in the data as well as responding to any control messages that give its name as the subject.

Jason agents are able to interact with a vehicle via the BSF, through the use of customised Jason environment and agent classes. The environment class has been extended, and uses the BSF sensor component with a subscription to spatial data. Received spatial updates are then processed within the environment class, and added as percepts to the relevant coordinator agent for a vehicle. The agent class has been extended with two BSF sensor components, one responsible for sending vehicle commands and one for sending data relating to the state of the Jason agents themselves. The latter is used to display information in the 3D view such as the number of beliefs and agent messages, in order to assist with identifying underlying reasons for observed behaviour of the vehicles. The agent class provides custom actions to the coordinator agent, where the two core actions (`setSpeed` and `setOrientation`) required for vehicle control are implemented. The `setSpeed` action is used by the coordinator agent to request that the vehicle moves at the specified speed, and the class extension passes this to its BSF sensor component, which constructs the appropriate RDF structure for XMPP transmission. The `setOrientation` action follows a similar process, generating a message for the BSF sensor, specifying the desired orientation of the vehicle. The overall process is very similar to that shown in Figure 3, apart from the delivery from the BSF is to vehicles instead of bots.

In this part of the study, the vehicle controlled by the Jason agents is simulated based on the implementation discussed earlier. On receipt of any vehicle control messages via the BSF, the simulated vehicle updates its speed or orientation value, which then takes effect during the next simulation step of the vehicle, as

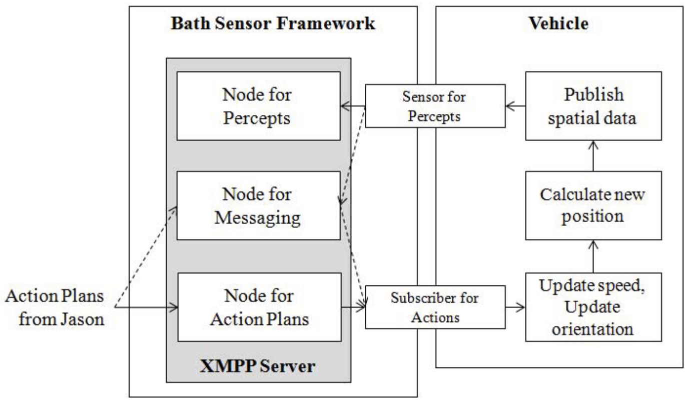

Fig. 5. AGAVE BSF vehicle integration

position changes are determined based on these values. This integration is shown in Figure 5, showing the communication flow between the BSF and the vehicle component.

Each vehicle is controlled by three agents: (i) a coordinator agent that acts as a gateway between other agents responsible for the vehicle and the interface to the vehicle itself – i.e. only this agent directly controls the vehicle), (ii) a driver agent responsible for achieving goals such as moving to a destination, and (iii) a convoy agent responsible for managing vehicle behaviour in convoy formations. Key plans provided by coordinator agent are `+!chosenSpeed(V)` and `+!requestTurnToAngle(A)`. The driver agent provides plans such as `+!emergencyStop`, `+!arrivedAtDestination`, `+!cruise`, `+!speedUp`, `+!slowDown` and `+!moveToKnownPosition` (which is dependent on `desiredXZ(X,Z)`). Finally, the convoy agent provides control largely based on belief updates about the car it is following, which in turn leads to requests for actions by the driver agent such as `desiredXZ(X,Z)` (which X,Z is the location of the vehicle being followed) and `speedUp` or `slowDown` to maintain convoy spacing.

This implementation has been used successfully for a scenario involving a convoy of vehicles navigating in the city centre of Bath (UK). Real world map data in OpenStreetMap format is used for this scenario, with a 3D virtual map created in order to view the simulated vehicles as they traverse the route.

AGAVE, Jason, and Bath Sensor Framework Vehicles. The objective here is to set up a bridge to a simple robot vehicle, such as a Lego Mindstorms, in place of the simulated vehicle used in the previous scenario. At present, both the vehicle control on the Jason side and on the Mindstorms platform are complete, but have not been tested together. In the case of the latter, we utilise an android phone for communications, which runs the BSF component to connect with the XMPP server and bluetooth commincations to send commands to the Mindstorms controller.

As intended, the system structure remains largely the same, due to the decoupled nature of the design as outlined earlier. The robot vehicle interface to the BSF only needs to implement the same functions as the simulated vehicle (i.e. `setSpeed` and `setOrientation`) and by doing so, the simulated vehicle and robot vehicle components become interchangeable, with no changes required to the Jason agents. Consequently, the same process as shown earlier in Figure 5 is used, however the calculation of the new position of the vehicle needs to be based on real world sensors rather than inferred from speed and orientation. This complication however resides with robot vehicle, but brings the benefit of being able to validate Jason agent performance against real sensor data and physical performance issues. As there are no changes required to the BSF design, Jason integration, or the Jason agents themselves, contrasting simulated with real results provides a useful comparison.

3.3 Evaluation

We have prototyped two demonstrators using the BSF: Jason agents controlling Second Life avatars, Jason agents controlling vehicles in a virtual driving environment. A third connecting Jason agents and Lego Mindstorms vehicles is almost completed. At the outset, our informal requirements were stated as (i) accessibility: connection of new software components (ii) performance: decoupling without significant degradation (iii) distribution: connection of components where-ever they might be, and (iv) scalability: in terms of size of environment and number of participants.

Clearly, at this stage, progress on scalability is not feasible, but we can comment on each of the other aspects, although we devote the most space to performance because seems to be the one that raises the most questions.

Accessibility. While each case study started out with the decoupling of the decision-making components (BDI agent) from the virtual environment, each has added other components that demonstrate the practice of our accessibility requirement.

In the context of the first case study, connection with the VE presented a challenge, because the OpenMetaverse library is written in C# and runs in .NET, hence required the construction of a C# client for the BSF, as well as cross-platform communications. Subsequently, we have incorporated a connection with an institutional model (involving Java and Answer Set Programming), following the initial implementation of Balke et al. [3], but decoupled by means of the BSF interface. This has been used to demonstrate norm-mediated behaviour of agents in Second Life (the “hello” example described earlier, and a more complicated one involving making space in a queue for an individual who is given priority).

The development history of the second case study, equally, illustrates how we are starting to meet this requirement. The system started as a decoupled version of the TankCoders [21] system, but the VE has been replaced by the jMonkeyEngine – a 3D game engine written in Java – and subsequently augmented by a 3D viewer that utilises Open Street Map data to produce a more visually credible environment for the convoy than the desert of the original TankCoders. The institutional connection will shortly also be applied here, as we start to treat the management of the convoy as a norm-governed environment problem.

Performance. A qualitative measure of performance might be that the system is performing properly only as long as no events are dropped. That would require even extreme situations to be withing the performance envelope. Quantitative evaluations may not be particularly helpful except perhaps to provide reasssurance that throughput of particular components is probably sufficient not to be the cause of a bottleneck. Even then, it may be hard to say, even if many such components are performing "well", whether their collective performance is adequate. We have, for example, measured round-trip message times between Second Life and the agent controlling an avatar, but this may well say more about the networking infrastructure than about the architecture as whole. As a consequence, it is common to eschew distribution for tight coupling in order to be able to deliver performance guarantees. Thus, performance is less about raw processing power, however that might be measured, but whether the architecture as a whole performs believably. Even then, exhaustive testing all possible states of all possible components is likely to be infeasible, so we are limited, as a poor substitute, to stress-testing individual components as a way of seeking confidence in the overall architecture. In practice, performance is a pervasive issue and tight coupling of components is one way that some control can be exerted over the collective factors that influence it, but in the long term such coupling impacts scalability.

It is too early to have detailed performance profiles of the components in the architecture we have described, but among the primary (new) sources of delay are the network, which is relatively hard to control, and the XMPP server (or servers, since they may be federated). There are several XMPP server implementations, but it is nor surprising, given the application domain, that all aim for high performance within themselves. We have chosen to use Openfire, because of its stated aims of supporting real time communications projects. We have not done a comparative evaluation against other XMPP servers. Openfire claims to be able to support significant numbers of (human) users with relatively few resources (e.g up 500 concurrent users with a minimum of 384Mb RAM on a 1.5GHz processor and up to 100,000 with a minimum of 2.0Gb RAM, 2$\times$3GHz processors and 1–4 connection managers). Further details at [38] measure factors such as the number of concurrent sessions and packet counts.

Since the fundamental mode of communication is publish/subscribe, one approach to evaluating the processing capacity of a component is to quantify the rate at which it can process incoming items, that is the data in the streams to which it is subscribed. Different components will have different subscription capacities, and depending on their role and where they are connected into the subscription network, one of several approaches may be appropriate if this capacity is insufficient, such as: (i) increasing component input capacity (ii) component replication (iii) throttling input volume, and (iv) inserting an aggregator

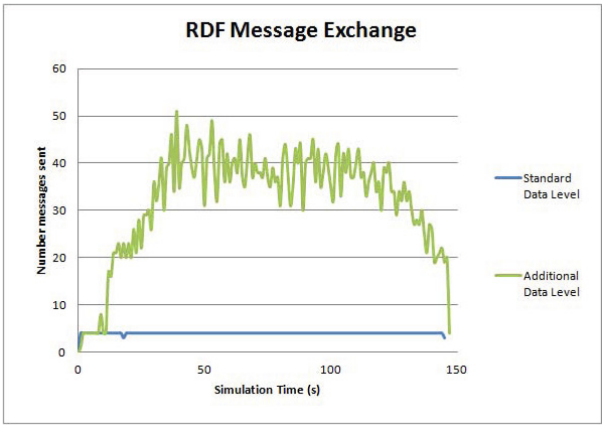

Fig. 6. Effect of additional jason state messaging on RDF quantity

component whose subscription and publication rates match upstream and downstream components. We have some preliminary data about subscription capacity, such as the frequency at which driver updates (in AGAVE) result in a stable simulation or lead to failure. For example, see Figure 6, where the Standard Data Level is the volume for normal convoy control, resulting in successful arrival at the destination, but the Additional Data Level, which includes full Jason state data, can result in a loss of communication between Jason agents and vehicles, depending on the number of vehicles involved. Openfire developers report of a capacity for ≈250 updates/sec – well in excess of the number observed above – but much more detailed measurements are required to fill out this picture.

There are two forms of mitigation that are possible in the architecture we have outlined: (i) short-circuiting, and (ii) aggregation (as mentioned earlier). Short circuiting is, in effect, taking events from the virtual environment, intercepting them before they are forwarded to the controlling agent, and making a decision that is returned to the VE. We do this, for example, in the Mindstorms scenario, where an android handset is physically located on the robot, which may make some (reactive) control decisions and relay them back over the local bluetooth connection to the (lower level) Mindstorms controller. Several such (nested) feedback loops [37] can be inserted into the control chain depending on need. Such a design pattern reflects a hierarchical control framework, where proximity to source implies lower level events and tighter control, as seen in historical multi-layer agent architectures such as InteRRaP [29] and Touring Machines [17]. The technical difference here is that those layers are distributed, reflecting the network-determined (or estimated) capacity for a timely response.

Aggregation is a complementary perspective on the same issue. Our experience and that of others [32] is that the Jason agents cannot handle high percept update frequencies (actual figures are not very useful because they are inevitably application and platform specific useful), which is typically manifested by unstable and hard to re-produce behaviour. One approach might be to re-engineer

Jason for higher performance, which although possibly desirable, does not consider whether all those events actually need to be processed at the BDI, that is the deliberative, level. In both theory and practice, cognitive architectures use layers to aggregate *small* observations into *bigger* ones. This can be characterised as inference or situational awareness, depending on perspective, but the overall effect is that minor observations are somehow collected, correlated and classified into less minor observations, subject to some degree of probability that reflects the accuracy of the process. In doing so, the volume of data, which possibly at some level may be labelled "information", is reduced so that frequency of communication is also reduced and the receiving reasoning process is presented with synthesized knowledge reflecting some kind of summary of the situation, rather than having to carry out that process itself. It is a fundamental design challenge, perhaps reflecting the principle of so-called sliding autonomy, to decide which levels should make which decisions, whether those strata are fixed and if not, how those divisions may be determined, or negotiated, in live situations.

We believe that performance is a many-faceted issue in this context and XMPP server throughput, and to a lesser extend network latency, while significant, are not the only factors, and it is as much the other components, but especially the deliberative architectures that we choose to use, the rate at which they can absorb percepts and the rate at which they can make effective decisions. This in turn is significantly affected by the level at which it is demanded they reason: is it sensible to use a BDI agent to monitor and adjust the speed of a vehicle –rather like a cruise control – when the same function can be achieved with a simple numerical procedure? Thus, the second mitigation is the relative ease with which new event processors can be added to this architecture, by subscribing them to existing feeds and publishing their results to existing consumers, through which it becomes possible to balance the factors of event rate, information level and network latency to achieve performance targets.

Distribution. Observations regarding distribution are relatively brief because, like accessibility, it could be viewed as having been demonstrated in principle, but like scalability, more is needed for it to be demonstrated with confidence. Since the XMPP message transport layer is directly built on HTTP, and since XMPP has been used for some years to support Internet Messaging in various guises (Microsoft Messenger, Google Talk, etc.), the mechanism has been demonstrated both to distribute and to scale. We have used the BSF in the context of a distributed sensing project, but the case studies reported here have only been run in the same local area network.

4 Related Work

AI research has paid substantial attention to how agent behaviour should reflect a response to something sensed from the environment in which it is situated. Cognitive architectures analyse this information, make decisions, and carry out planning to determine the next behaviour to execute. In a dynamic environments, SOAR [24] is a well known example of a classical symbolic reasoning architecture. However, it is also known as rather heavy-weight and can hardly be expected to respond in real time. There is also a range of well-known reactive architectures, including subsumption [13], Finite State Machines (FSMs) [12, 14], Basic Reactive Plans (BRP) [14, 18], and POSH plans [11], amongst others. Any of the above, perhaps bar SOAR, are suitable decision-makers for avatars in virtual environments, but our choice from among goal-driven approaches, is the popular the Belief-Desire-Intention (BDI) architecture [33]. Beliefs here refer to knowledge about the world in agents mind, desires are objectives to be achieved, and intentions identify the actions chosen by the agent as part of some plan to help it achieve a particular desire [28].

A distinct line of research has highlighted the notion of the environment programming in multiagent systems [36]. According to [36], agent programming should have balance between the agent itself and its environments in order to achieve a high level of intelligence. This perspective reflects the idea that the environment becomes a meaningful place to support the agent's abilities with many functionalities, rather than the traditional view in which it is simply a place that the agent senses and acts upon.

In this context, there is a fair body of research into the deployment of an embodied artificial intelligence using the above cognitive architectures in rich environments. For example, Bogdanovych et al [7] introduce the 3D Electronic Institution, or Virtual Institution (VI), which is a virtual world with normative regulations governing interactions between participants and environment. They also also propose the introduction of virtual characters capable of learning [8] and the use of VI as environments for imitation learning, providing for the enhancement of virtual agent behaviour by learning from human users or other software agents. Later work from this group puts forward a teaching mechanism, so that the virtual character may become more believable [9]. Although this work is amongst the most developed in the use of Second Life, it offers rather less on the matter of agent-environment programming and the role of cognitive architectures, because of its focus on the regimented normative environment and virtual characters that learn.

Veksler [42] demonstrates integration between ACT-R [2, 23] and Second Life via HTTP web server. In this work, all information are gathered by a 3D object, which is attached to the avatar. It scans the environment around the avatar within a certain radius, and sends what scanner senses to a dedicated web server. The ACT-R module is separate from the Second Life environment, but capable of communicating to the web server. By means of HTTP request to the web server, the decision-making module collects sensing data and executes a 'perceive–think–act' loop. In the end, the decision including motor actions goes back to the intermediate web server, and are then applied to the avatar. Another notable work is [31], in which a Jason agent supplies the reasoning for a virtual agent in an environment provided by Second Life, supported by an external data processing module that handles environment sensing. In the same manner as above, through an attached 3D object, which serves as a virtual sensory system, sets of perceptions generated by the data processing module are delivered to Jason agent, which then deliberates. The results of the reasoning are communicated back to the Second Life avatar, and the action is realised, changing the state of the environment. The scenario in this case is the playing of a football game. This work demonstrates the utilization of an event recognition platform [32] not only to enhance the perception capability, which becomes a source of better reasoning, but also to retrieve more accurate domain-specific information from low level data.

In comparison with the above systems, which are quite tightly coupled, other approaches also exist, that aim for a more general integration between cognitive agents and virtual environments. There are (at least) three representative systems, with similar objectives to ours, against which we contrast what has been presented here: GameBots [1], Pogamut [22], and CIGA [41]. Gamebots [1] has much in common with the virtual agent component in our system, being a kind of programmable agent controller, integrated with the 3D video game Unreal Tournament (UT), in order to create autonomous bots that interact with human players as well as other bot players. The bots are able to sense and act directly from the environment, via TCP/IP socket communication. Gamebot 'agents' appear to be limited to reactive behaviours, while the system as a whole only functions with one game engine, namely Unreal Tournament.

Pogamut [22] incorporates an interface layer between Unreal Tournament and the decision-making agent, by means of TCP/IP sockets. The role of this component is rather like that of the Jason agent in our system, in that it has the task of perceiving the environment, interaction with environment, and decision making, for which it uses the POSH reactive planner [11]. As with Gamebots, Pogamut has seen substantial up-take, from student projects to complex research projects, thanks to their approach that allows greater flexibility in the development of the high level of autonomy in virtual agents. Nevertheless, it still has a high dependency on the particular environment of UT, and on a particular programming language, namely Java.

CIGA [41] also has numerous similarities with our framework in that it aims to resolve the coupling problem between agent and virtual environment. This it does by means of two interface layers and an ontology model: (i) physical interface layer to connect to a environment (game engine), and (ii) cognitive interface layer to connect to a multiagent system, corresponding to the Virtual Agent and the Jason Agent, respectively. The use of ontology model, containing pre-defined ontologies to make a contract between agent and game engine even though they are situated in a specific domain, eases the interpretation of perception and behaviour execution. This architecture offers fair accessibility, and could in principle support distributed execution, thanks to the use of socket-based communications, but this would require careful manual configuration. In this respect, CIGA is the closest to our proposal, but the dependence on a relatively low-level and inflexible network layer seems likely to inhibit distribution and scalability.

To summarise, the short-comings we observe in the above lie in their tight integration of the components, leading to an effectively closed, single platform system. Thus, they do not have the flexibility necessary for distributed software systems. They are tightly coupled by the communication protocol as well as ontology, so that adding/removing a software component is challenging, and deploying such platforms more widely is in general difficult. This is only exacerbated – or probably even rendered impossible – if it is desired to incorporate components in different programming languages or that only run on another operating system.

Earlier, we noted the lack of any comprehensive performance evaluation of XMPP, as far as we can find, in the academic literature. Linden Labs have apparently carried out an evaluation of various message passing protocols[2], noting that the Advanced Message Queueing Protocol [15] implementations demonstrate good single-host performance, but lack figures on maximum capacity when clustered, or the value of clustering. XMPP appears in the list, but was not evaluated for lack of time. Several other protocols are eliminated for not meeting their requirements, but unfortunately there is no definitive conclusion. At best, this indicates that if XMPP proves inadequate, AMPQ may be worth investigation, however it is observed that AMPQ lacks adequate flow control mechanisms, so servers may simply stop when overloaded, unlike XMPP. Also, we note that communication may well finally constitute only a small fraction of the critical costs of round-trip times between agent and VE, compared to the response times of the agent or the VE themselves.

5 Conclusion and Future Work

In this paper, the Bath Sensor Framework has been introduced as a middleware for decoupling cognitive agents and virtual environments. Also, two case studies are presented using the framework for linking heterogeneous software, the Jason BDI reasoning platform, and Second Life and AGAVE which are representative rich 3D virtual environments, respectively. From these studies, it seems clear that the BSF has some useful advantages as an integration middleware. Firstly, it offers good accessibility, because of the simplicity in protocol, and ease of both connection and use. Secondly, in respect of speed and reliability, it inherits from XMPP, so that it is able not only to communicate in real time but also transfer whatever data in the form of open XML, which may become the basis of the interoperability in cross domain applications. Finally, it enables distribution of components, so that it contributes to effective data transport mechanism such as 1-to-1, 1-to-many, or many-to-many, from anywhere to anywhere. As a result, through the use of the BSF, the system as a whole has the potential for flexibility and extensibility.

Furthermore, we have extended the framework to operate in conjunction with an institutional framework based, so that the behaviour of agent in virtual environments can be governed by norms.The notion of institution, as a set of rules for a particular agent society, is appropriate for the regulation of behaviour of

[2] `http://wiki.secondlife.com/wiki/Message_Queue_Evaluation_Notes`, retrieved 20120416, last updated 2010.

virtual agents in a particular situations, by saving the need to incorporate behaviour for all circumstances in the agent themselves.

Plans for future work include a careful evaluation of performance issues, such as (i) the notion of component subscription profiles, (ii) monitoring of communication in live systems, so that out-of-profile situations can be detected, (iii) development of mitigations, such as throttling, replication and aggregation, (iv) experimentation with data handling policies, such as discarding and (finite) buffering, amongst others and (v) exploring the feasibility of applying corrective actions during execution, as well, of course, as the application of IVAs to more demanding scenarios.

Acknowledgements. We would like to thank Surangika Ranathunga, Stephen Cranefield, and Martin Purvis for useful discussions and observations.

References

1. Adobbati, R., Marshall, A.N., Scholer, A., Tejada, S.: Gamebots: A 3d virtual world test-bed for multi-agent research. In: Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS (2001)
2. Anderson, J.R., Matessa, M., Lebiere, C.: ACT-R: A theory of higher level cognition and its relation to visual attention. Human Computer Interaction 12(4), 439–462 (1997)
3. Balke, T., De Vos, M., Padget, J., Traskas, D.: On-line reasoning for institutionally-situated bdi agents. In: The 10th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2011, Richland, SC, vol. 3, pp. 1109–1110. International Foundation for Autonomous Agents and Multiagent Systems (2011)
4. Bergenhem, C., Huang, Q., Benmimoun, A., Robinson, T.: Challenges of platooning on public motorways. In: 17th World Congress on Intelligent Transport Systems (2010), `http://www.sartre-project.eu/en/publications/Documents/ITS%20WC%20challenges%20of%20platooning%20concept%20and%20modelling%2010%20b.pdf` (retrieved November 11, 2012)
5. Bernstein, D., Vij, D.: Intercloud directory and exchange protocol detail using XMPP and RDF. In: 2010 6th World Congress on Services (SERVICES-1), pp. 431–438 (July 2010)
6. Bernstein, D., Vij, D.: Using XMPP as a transport in intercloud protocols. In: 2010 the 2nd International Conference on Cloud Computing, CloudComp (2010)
7. Bogdanovych, A., Esteva, M., Simoff, S., Sierra, C., Berger, H.: A Methodology for Developing Multiagent Systems as 3D Electronic Institutions. In: Luck, M., Padgham, L. (eds.) Agent-Oriented Software Engineering VIII. LNCS, vol. 4951, pp. 103–117. Springer, Heidelberg (2008)
8. Bogdanovych, A., Simoff, S., Esteva, M.: Virtual Institutions: Normative Environments Facilitating Imitation Learning in Virtual Agents. In: Prendinger, H., Lester, J.C., Ishizuka, M. (eds.) IVA 2008. LNCS (LNAI), vol. 5208, pp. 456–464. Springer, Heidelberg (2008)
9. Bogdanovych, A., Simoff, S., Esteva, M., Debenham, J.: Teaching autonomous agents to move in a believable manner within virtual institutions. In: Bramer, M. (ed.) Artificial Intelligence in Theory and Practice II. IFIP, vol. 276, pp. 55–64. Springer, Heidelberg (2008)

10. Bordini, R.H., Wooldridge, M., Hübner, J.F.: Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology). John Wiley & Sons (2007)
11. Brom, C., Bryson, J.J.: Action selection for intelligent systems. In: The European Network for the Advancement of Artificial Cognitive Systems, white paper 044-1 (2006)
12. Brooks, R.A.: Intelligence without representation. Artificial Intelligence 47(1-3), 139–159 (1991)
13. Brooks, R.A.: How to build complete creatures rather than isolated cognitive simulators. In: Architectures for Intelligence. Lawrence Erlbaum Assosiates, Mahwah (2001)
14. Bryson, J.J.: Action selection and individuation in agent based modelling. In: Proceedings of AGENT 2003: Challenges of Social Simulation, pp. 317–330 (2003)
15. OASIS Advanced Message Queueing Protocol (AMQP) Technical Committee. Advanced message queuing protocol 1.0. Technical report, OASIS (2012), `https://www.amqp.org/resources/download` (retrieved 20120416)
16. Dressler, F., Kargl, F., Ott, J., Tonguz, O., Wischhof, L.: 10402 abstracts collection and executive summary – inter-vehicular communication. In: Inter-Vehicular Communication, Dagstuhl, Germany. Dagstuhl Seminar Proceedings, vol. 10402, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2011)
17. Ferguson, I.A.: Touring machines: Autonomous agents with attitudes. Computer 25(5), 51–55 (1992)
18. Fikes, R.E., Hart, P.E., Nilsson, N.J.: Learning and executing generalized robot plans. Artificial Intelligence 3, 251–288 (1972)
19. The XMPP Standards Foundation. Extensible messaging and presence protocol(XMPP): Core, and related other RFCs. `http://xmpp.org/rfcs/rfc3920.html`
20. The XMPP Standards Foundation. The XMPP standard foundation homepage. `http://www.xmpp.org`
21. Fronza, G.: Simulador de um ambiente virtual distribuido multiusuario para batalhas de tanques 3d com inteligencia baseada em agentes BDI. Final year project report (July 2008),`http://campeche.inf.furb.br/tccs/2008-I/2008-1-14-ap-germanofronza.pdf`, See also `http://sourceforge.net/projects/tankcoders/` (retrieved Novebber 11, 2012)
22. Gemrot, J., Kadlec, R., Bída, M., Burkert, O., Píbil, R., Havlíček, J., Zemčák, L., Šimlovič, J., Vansa, R., Štolba, M., Plch, T., Brom, C.: Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents. In: Dignum, F., Bradshaw, J., Silverman, B., van Doesburg, W. (eds.) Agents for Games and Simulations. LNCS, vol. 5920, pp. 1–15. Springer, Heidelberg (2009)
23. ACT-R Research Group. ACT-R: Theory and architecture of cognition, `http://act-r.psy.cmu.edu/`
24. The Soar Group. Soar project homepage, `http://sitemaker.umich.edu/soar`
25. Jabber-Net. The jabber.net project, `http://code.google.com/p/jabber-net`
26. Kumar, S., Chhugani, J., Kim, C., Kim, D., Nguyen, A., Dubey, P., Bienia, C., Kim, Y.: Second life and the new generation of virtual worlds. Computer 41(9), 46–53 (2008)
27. Linden Labs. Second life homepage, `http://www.secondlife.com`
28. Mascardi, V., Demergasso, D., Ancona, D.: Languages for programming bdi-style agents: an overview. In: Corradini, F., De Paoli, F., Merelli, E., Omicini, A. (eds.) WOA, pp. 9–15. Pitagora Editrice Bologna (2005)
29. Müller, J.: The Agent Architecture InteRRaP. In: Müller, J.P. (ed.) The Design of Intelligent Agents. LNCS, vol. 1177, pp. 45–123. Springer, Heidelberg (1996)

30. OpenMetaverse Organization. libopenmetaverse developer wiki, `http://lib.openmetaverse.org/wiki/`
31. Ranathunga, S., Cranefield, S., Purvis, M.: Interfacing a cognitive agent platform with a virtual world: a case study using second life. In: The 10th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2011, vol. 3, pp. 1181–1182. International Foundation for Autonomous Agents and Multiagent Systems, Richland (2011)
32. Ranathunga, S., Cranefield, S., Purvis, M.: Identifying events taking place in second life virtual environments. Applied Artificial Intelligence 26(1-2), 137–181 (2012)
33. Rao, A.S., Georgeff, M.P.: BDI agents: from theory to practice. In: Proceedings of the First Intl. Conference on Multiagent Systems, San Francisco (1995)
34. Ignite Realtime. The Openfire Project, `http://www.igniterealtime.org/projects/openfire/`
35. Ignite Realtime. The Smack API Project, `http://www.igniterealtime.org/projects/smack/`
36. Ricci, A., Piunti, M., Viroli, M.: Environment programming in MAS: An artifact-based perspective. Autonomous Agents and MultiAgent Systems 23(2), 158–192 (2011)
37. Van Roy, P.: Self management and the future of software design. Electr. Notes Theor. Comput. Sci. 182, 201–217 (2007)
38. Jive Software. Openfire scalability, `http://www.igniterealtime.org/about/OpenfireScalability.pdf` (retrieved November 09, 2012)
39. Stout, L., Murphy, M.A., Goasguen, S.: Kestrel: an XMPP-based framework for many task computing applications. In: Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, MTAGS 2009, pp. 11:1–11:6. ACM, New York (2009)
40. In-Band Real Time Text. Xep-301: In-band real time text. Technical report, XMPP Standards Foundation (2012) `http://xmpp.org/extensions/xep-0301.pdf` (retrieved April 16, 2012)
41. van Oijen, J., Vanhée, L., Dignum, F.: CIGA: A Middleware for Intelligent Agents in Virtual Environments. In: Proceedings of the 3rd International Workshop on the uses of Agents for Education, Games and Simulations (2011)
42. Veksler, V.D.: Second-life as a simulation environment: Rich, high-fidelity world, minus the hassles. In: Proceedings of the 9th International Conference of Cognitive Modeling (2009)
43. Wagener, J., Spjuth, O., Willighagen, E., Wikberg, J.: XMPP for cloud computing in bioinformatics supporting discovery and invocation of asynchronous web services. BMC Bioinformatics 10(1), 279 (2009)