Cross-Domain Interoperability: a Case Study

Jukka Honkola, Hannu Laine, Ronald Brown, and Ian Oliver

Nokia Research Center
P.O. Box 407
FI-00045 NOKIA GROUP
jukka.honkola@nokia.com, hannu.e.laine@nokia.com
ronald.brown@nokia.com, ian.oliver@nokia.com

Abstract. We describe a case study of the behaviour of four agents using a space based communication architecture. We demonstrate that interoperability may be achieved by the agents merely describing information about themselves using an agreed upon common ontology. The case study consists of an exercise logger, a game, a mood rendered attached to audio player and phone status observing agents. The desired scenario emerges from the independent actions of the agents

1 Introduction

Interoperability between devices and the software and applications that run on those devices is probably the major goal for many developers of such systems. This can be achieved in a number of ways: particularly through open, agreed standards or often through monopoly. Whichever particular 'interoperability' route is taken there will always be a plurality of standards relating to how such systems communicate [6][1].

For example, the Universal Plug and Play interoperability standard is plagued with manufacturer and device specific variations which complicate and nullify to some extent the long and complex standardization process employed to avoid this.

Technologies such as the Semantic Web [2] provide enablers for solutions to these problems. Within the Semantic Web exist information representation formats such as the Resource Description Framework (RDF) [10] and the web ontology language OWL [9] which themselves build upon enablers such as XML. Using these technologies we can address and provide solutions[1] to the interoperability problem.

At one level there are existing solutions such as Web Services (SOAP, WSDL, UDDI etc) and various stacks of middleware for processing representation formats and preservation and reasoning about the semantics of messages. At the other, there are solutions based around more declarative mechanisms such as TripCom [11] and the space-based solutions, for example Java Spaces [4] for the transport and processing of messages and information to assist in interoperability.

We do not believe that interoperability will be achieved through standardization committees, nor through standard, globally accepted semantics and ontologies but rather by the unification of semantics in a localized manner, as described in previous work [8].

[1] plural, never singular!

In this paper we describe an approach to solving the interoperability problem through a combination of context gathering, reasoning and agents within a space-based infrastructure taking advantage of technologies such as RDF. We demonstrate this through the ad hoc integration or mash-up of a number of distinct applications to demonstrate the principles of this approach.

2 The M3 Concept

The M3 system consists of a space based communication mechanism for independent agents. The agents communicate implicitly by inserting information to the space and querying the information in the space. The space is represented by one or more semantic information brokers (SIBs), which store the information as an RDF graph. The agents can access the space by connecting to any of the SIBs making up the space by whatever connectivity mechanims the SIBs offer. Usually, the connection will be over some network, and the agents will be running on various devices. The information in the space is the union of the information contained in the participating SIBs. Thus, the agent sees the same information content regardless of the SIB to which it is connected. The high-level system architecture is shown in Figure 1.

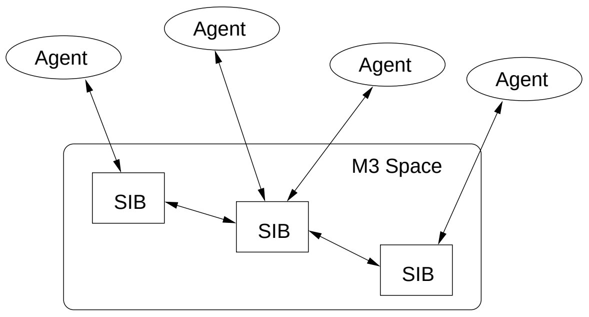

Fig. 1. A diagram of the general system architecture: Agents, M3 spaces and SIBs.

The agents may use five different operations to access the information stored in the space:

Insert	: Insert information in the space
Remove	: Remove information from the space
Update	: Atomically update the information, i.e. a combination of insert and remove executed atomically
Query	: Query for information in the space
Subscribe	: Set up a persistent query in the space; changes to the query results are reported to the subscriber

In addition to the four access operations there are *Join* and *Leave* operations. An agent must have *joined* the space in order to access the information in the space. The join and leave operations can thus be used to provide access control and encrypted sessions, though the exact mechanisms for these are still undefined.

In its basic form the M3 space does not restrict the structure or semantics of the information in any way. Thus, we do not enforce nor guarantee adherence to any specific ontologies, neither do we provide any complex reasoning[2]. Furthermore, information consistency is not guaranteed. The agents accessing the space are free to interpret the information in whatever way they want.

We are planning to provide, though, a mechanism to attach agents directly to the SIBs. These agents have a more powerful interface to access the information and can be e.g. guaranteed exclusive access to the information for series of operations. Such agents may perform more complex reasoning, for example ontology repair or translation between different ontologies. However, they may not join any other spaces but are fixed to a single SIB and thus a single space.

The M3 spaces are of local and dynamic nature, in contrast to semantic web which embodies Tim Berners-Lee's idea of semantic web [2] as a "giant global graph". The locality and dynamicity—we envision that the spaces will store very dynamic context information, for example—poses different challenges than the internet-wide semantic web. For example, in order to provide a true interoperability for local ubiquitous agents, the space (i.e. SIBs) will have to provide a multitude of connectivity options in addition to http: plain TCP/IP, NoTA [7], Bluetooth,. . . Furthermore, the space should be fairly responsive. While we do not aim for real-time or near real-time system, even half minute long response times for operations are unacceptable.

The responsiveness is one of the factors behind the fundamental decision to not enforce any specific ontologies and allowing the agents to interpret the information freely, as it lessens the computational burden of the infrastructure. Another, and more important reason is that we explicitly want to allow mashing up information from different domains in whatever way the agents see best. Strict ontology enforcement would make this kind of activity extremely difficult as all new ways of mashing up the information would require approval from some ontology governance committee. However, as mentioned above, we still plan to provide means for ontology enforcement for cases where the space provider explicitly wishes to restrict the ways the information is used as there are bound to be also situations where this is the best approach.

The information content in a M3 space may be distributed over several SIBs. The distribution mechanism assumes that the set of SIBs forming a M3 space are totally routable but not necessarily totally connected. The information content that the agents see is the same regardless the SIB where they are connected.

2.1 Applications in M3 Spaces

The notion of application in M3 space is differs radically from the traditional notion of a monolithic application. Rather, as a long term vision, we see the applications as possible scenarios which are enabled by certain sets of agents. Thus, we do not see an

[2] The current implementation of the concept understands the owl:sameAs concept

email application running in M3 space, but we could have a collection of agents present which allow for sending, receiving, composing and reading email.

For this kind of scenario based notion of application, we also would like to know whether the available agents can succesfully execute the scenario. The envisioned model of using this system is that the user has a set of agents which are capable of executing certain scenarios. If a user needs to perform a new scenario that the current set of agents are not capable of executing, she could go and find a suitable agent from some directory by describing the desired scenario and the agents she already has.

Thus, we need some formal or semi-formal way of describing agent behavior both with respect to the M3 space and to the environment. While there exists research addressing behavior in multi-agent systems, for example by Herlea, Jonker, Treur and Wijngaards [5], this kind of ad-hoc assembly of agents in order to execute a certain scenario seems to be quite unaddressed in current research. However, slightly similar problems have been addressed in e.g. web service orchestration research [3], but these still seem to concentrate on design-time analysis rather than run-time analysis.

As for shorter term, our vision is that sets of existing applications would be enhanced by being able to interoperate and thus allow execution of (automatic) scenarios that would have been impossible or required extensive work to implement without the M3 approach.

3 The Case Study

The case study consists of four agents, Exercise logger, SuperTux game, Phone line observer, and Mood renderer, running on several devices. The exercise logger and phone line observer run on Nokia S60 platform phones, and the SuperTux and mood renderer run on Nokia N800 internet tablets. The M3 space infrastructure runs on a Linux laptop.

3.1 Application Development Vision

The case study setup illustrates a "legacy enhancement" approach to application development in M3 spaces. Conceptually, we added M3 space capability to existing applications (SuperTux, Mood renderer, SportsTracker, Telephony) even if the SportsTracker and Telephony were implemented as standalone simple applications for practical reasons.

The starting point for the case study was a single person (as defined in Person ontology, section 3.2) who would be exercising and tracking the workouts with an exercise logger, playing computer games and listening to music. The goal was to demonstrate the benefits of interoperability between existing application, that is, to be able to execute a scenario where the SuperTux game would award extra lives for exercising, a mood renderer embedded in a media player would play suitable music depending on a game state, and the game and media player would react accordingly if the person receives a call.

3.2 Ontologies

The ontologies used in this case study have been defined informally, and are not enforced in any way. The different components of the case study assume that they are used correctly. A pseudo-UML diagram of the complete ontology is shown in Figure 2, and an example instantiation of the ontology when a game is being played and the phoneline is idle is shown in Figure 3.

As a general note, the needs of the scenario studied drove the modeling of the ontologies. Most of the choices of what to include and what to exclude were based on the need of the information in executing the chosen scenario. A broader study might need to revisit the ontologies and generalize end expand them.

The agents use different parts of the combined[3] case study ontology. The Workout Monitor agent understands Workout and Person ontologies, the SuperTux agent understands all ontologies, the Telephony Observer agent understands OperatorAccount and Person ontologies, and the Mood Renderer understands Game and OperatorAccount ontologies.

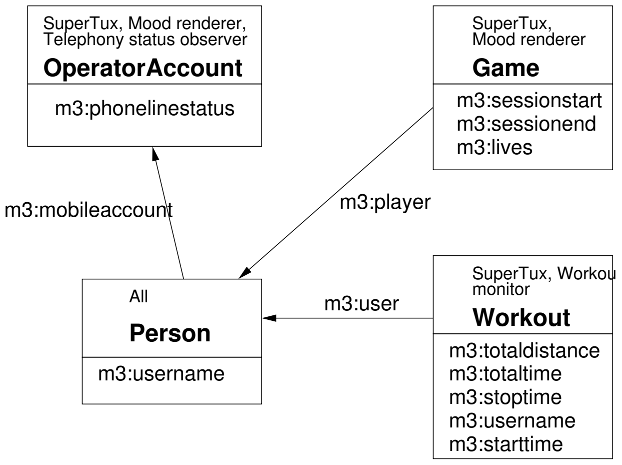

Fig. 2. An informal description of used ontologies. The using agents are named on top of the “class” box

Person The person ontology describes the information about a person necessary for this case study. A person can have a name (`m3:username`) and an operator account (`m3:mobileaccount`).

[3] A combination of Person, OperatorAccount, Game, and Workout ontologies

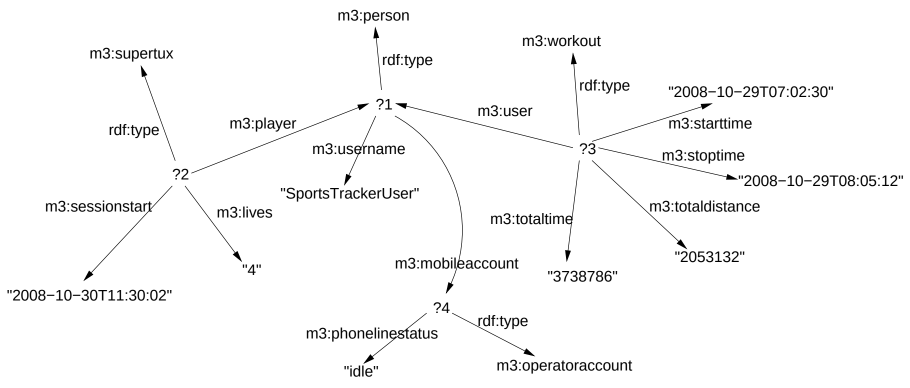

Fig. 3. An example RDF graph during the execution of scenario

Operator Account The operator account ontology contains information about the phoneline status of the account. In a more realistic case, there could also be information about the phone number, used talking time, amount of data transferred etc.

Workout The workout ontology describes a generic distance-oriented workout. Thus, we include information about the length of the workout, distance traveled, person who has done the workout etc. but no information related to, for example, weightlifting where we probably would like to know the number of repeats and weights instead of time used and distance. The choice of what to include in the ontology was influenced by the information available in the SportsTracker application.

Game The Supertux game ontology includes information about the gaming session start time, stop time, lives left and player. The session start is defined to be the time when the game executable was started, showing a welcome screen. The lives value is inserted when a player starts a game, new or saved, from the welcome screen. When the player quits the game completely, that is, exits the welcome screen, the session stop time is inserted.

The ontology could include also information about current score, amount of coins, field being played etc. which the mood renderer could use.

3.3 Agents

Next we describe the operation of agents making up the case study. It should be noted that the agents are only communicating implicitly through the M3 space by reading the information inserted to it by other agents.

Workout Monitor The workout monitor reads SportsTracker workout files and writes a summary of the workouts to M3 space. We currently read the information related to general description of the workout and insert that in the M3 space. Any path information is ignored as we have no use for it in the current scenarios.

The workout information is related to the person doing the workouts. If a person with same name as the SportsTracker username is found, we attach the workout in question to that person, otherwise we create a new person.

We use an SportsTracker internal workout identifier to identify workouts so that we do not insert duplicate workouts if the monitor is run again with same workouts as before.

Supertux Supertux is a classic jump'n'run sidescroller game which has taken strong inspiration from the original SuperMario games for Nintendo. Supertux is open source and available under GPL.

The ontologies related to Supertux game are the Person ontology, the Workout ontology and the Supertux ontology. Each Supertux game is assigned with player information. If the person class instance of the player can not be found from the M3 space the game creates one and inserts player information according to the Person ontology. The person class instance also binds the player to the workout information inserted to the M3 space. Prior starting the game supertux queries workout information from the M3 space. The first query searches for all instances of the Workout class. For each found workout class instance a new query is performed in order to find out the details of the workout. If there are long and recent enough workouts performed by the player the game awards two extra lives to a game level.

The game creates also an instance of itself and inserts information such as reference to the person class of the player, session start and stop time to the M3 space. In addition, the game updates how many lives the player has left during playing a game level. Every time the player looses or gains one life in the game, the game removes old `lives`information and inserts the new information to the M3 space.

Telephony status observer The telephony voice line status is part of a OperatorAccount class. Operator accounts are bound to a person *i.e.* it is assumed that each account belongs to a person[4]. The voice line status only has two states, namely `active` and `idle`. The phone line status monitor subscribes to voice line status changes provided by Symbian's CTelephony API. When the user dials a phone number to establish voice connection or when there is incoming call to the phone, CTelephony API provides the current voice line status to the observer application. The observer application transforms the CTelephony states (*e.g. EStatusRinging, EStatusDialling, EStatusIdle*, ...) to corresponding values defined by the OperatorAccount ontology and updates the information to the M3 space.

Mood renderer A mood renderer conceptually would have interfaces to sensory, i.e. mood rendering devices of a smart space invironment in order to cooridinate their controls according to a mood determined by its knowlege processing. The questions would be many about such knowlege for determing a mood and how to render it for a given

[4] or a group of persons—we do not prevent that

environment. The mood renderer of the case study is a very simple start on the problem, which side-steps many issues for practical cases, while showing the essence of knowlege processing for the given concept.

The mood renderer in this case study is based on music as the only rendering device, and thus, the simple assumption that music, more precisely some audio track, maps to a determinable mood, and still further, that the mapping stems solely from the state of a partical game being played and applies invariently with regard to time. This dodges questions about personal tastes and more than one person contributing to the smart space mood.

In choosing a track to play, volume is not controlled, but play, pause and stop are used where appropriate. Mood rendering is based solely on two sources of information in the smart space environment. The first is an operating supertux game and the second is phone status. Note then, that this mood renderer is solely reactive to information about a dynamic smart space environment to which it belongs—for simplicity, it does not contribute information even though it clearly impacts it.

Given the simplifications stated, and that the mood renderer understands the ontologies of the supertux game and telephony status observer. It follows their two independent contributions to the smart space environment and combines them to render a mood appropriate for the state of game play, while maintaining the courtesy of pausing an audio track, if being played while any phone status of the smart space is active.

The mood renderer is implemented as two persistent queries for smart space information. The first is looking for an active game in order to render its mood, and the second is looking for any phone with an active status. When an active game is found, a third persistent query, particular to this game, looks for its level of `m3:lives` and renders a mood as CALM, for no lives before the real play starts, and then when `m3:lives` exist, either “UP-BEAT” or “DOWN-BEAT” according to a threshold configuration on the number of lives. When the game ends, audio rendering is stopped.

The mood renderer’s behavior can be described as follows:

- Start with `renderer.setTrack (NONE)`, `renderer.setMode (STOP)`, and `gameSelected=FALSE`.
- Subscribe to the call-status attribute for all phone-status-observers. Monitor subscription results, for any attibute value of “active”, if `currentMode` is `PLAY`, then do `renderer.setMode (PAUSE)`; otherwise, if `currentMode` is `PAUSE`, do `renderer.setMode (PLAY)` *i.e.* any playing pauses during any phone call, or resumes when no phone call if paused during a game.
- Subscribe to a supertux game, which has a session-start and no session-end, or session-start after session-end. Monitor subscription results, and, whenever fulfilled, first ensure that rendering is stopped and end subscriptions for any earlier game. Then render music/mood mapping for the selected game using the next subscription.
- Subscribe to session-end and lives attributes of the selected game. When fulfilled, set `renderer.setMood (mood)`, for mood according to lives as follows:
  - no lives: CALM *i.e.* game started, but not yet played.
  - lives above threshold: UP-BEAT
  - below threshold: DOWN-BEAT

When session-end of game is after session-start, stop rendering and end subscriptions about this game.

4 Conclusions

The idea of agents interoperating by communicating implicitly through M3 spaces worked well for this case study. We were able to execute the scenario in mind and add the phone agent to it without modifying the existing ontology. However, this work was performed inside a single team in a single company. This kind of close cooperation between people implementing the agents obviously makes things easier.

Most of the work required in integrating the different agents into a coherent scenario was related to the mood renderer observing the state of the SuperTux game. The ontology did not directly provide information about all the state changes in the game. For example, when starting to play a level in the game, the mood renderer deduce this by observing the appearance of the `lives` attribute in the M3 space. This is however something that will probably be a very common case as the ontologies can not and should not try to describe all possible information that some agent may need. It should suffice that the information is deducible from the explicitly stated information.

Another aspect of local pervasive computing environments that was not really addressed in this case study was the handling of resources that can only be used by one agent at a time. In this case the mood renderer represents such resource (audio player), and implicitly it is usually the game that uses it as the mood renderer deduces the mood based on the game state alone. When the phone rings, the mood renderer will stop the music and continue after the call is finished. However, in a more complex case there needs to be (preferably) clear policy for each resource on who can use it. We currently think that this can be handled at the ontology level, with suitable coordination structures and an agent acting as a gatekeeper. In general, this is however a subject for further study.

The basic principles of agent-dependent interpretation of information and implicit communication proved useful when the phoneline status information was added to the case study. When we only describe the information and not the actions to be taken by the agents, the agent can select suitable course of action based on the detailed knowledge of the agent in question. In this particular case, the correct course of action for the SuperTux game is to pause when the phoneline is active but not automatically resume the game when it goes idle, as the player is not necessarly ready to continue immediately after ending the call. On the other hand, the mood renderer should continue playing the music immediately after the distraction (in this case call) has ended.

If we expanded the case study with more persons with each having mobile phones, the difference with mood renderer and SuperTux regarding the behavior when a call arrives would be even more. Sensible behavior for mood renderer would be to stop the music whenever anyone in the same (acoustic) location receives a call, while the game should only react to calls to the player of the game.

Of course, in some cases it might be desirable to have coordinated response from a group of agents to certain events. We believe that this is best handled outside the system, by e.g. requirements in the scenario descriptions or ontology documentation.

Anyhow, such coordinated responses would not apply to all agents but rather to only certain groups of agents involved in executing a given scenario.

An obvious next step related to the interoperability hypothesis is to have external research partners extend the demo with agents implemented outside Nokia. This would clarify the required level of ontology and agent behavior description as the communication would then approximate more the real world case where the agents would be implemented by multiple vendors.

5 Acknowledgements

The authors would like to thank Vesa Luukkala for helping with ontology modeling, and Antti Lappeteläinen for helpful comments.

References

1. S. Bergamaschi, S. Castano, and M. Vincini. Semantic Integration of Semi-structured and Structured Data Sources. *SIGMOD Record*, 28(1):54–59, March 1999.
2. Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, May 2001.
3. H. Foster, S. Uchitel, J. Magee, and J. Kramer. Compatibility verification for web service choreography. In *Proceedings of IEEE International Conference on Web Services, July 6–9 2004*, pages 738–741. IEEE, 2004.
4. Eric Freeman, Susanne Hupfer, and Ken Arnold. *JavaSpaces Principles, Patterns and Practice*. Addison-Wesley, 1999.
5. Daniela E. Herlea, Catholijn M. Jonker, Jan Treur, and Niek J. E. Wijngaards. *Multi-Agent System Engineering*, volume 1647 of *LNCS*, chapter Specification of Bahavioural Requirements within Compositional Multi-agent System Design, pages 8–27. Springer, 1999.
6. Grace A. Lewis, Edwin Morris, Soumya Simanta, and Lutz Wrage. Why Standards Are Not Enough To Guarantee End-to-End Interoperability. In *Proceedings of 7th IEEE International Conference on Composition-Based Software Systems (ICCBSS), February 25–29 2008, Madrid, Spain*. IEEE, 2008.
7. Network on terminal architecture. http://www.notaworld.org, 11 2008.
8. Ian Oliver and Jukka Honkola. Personal Semantic Web Through A Space Based Computing Environment. In *Second IEEE Interntional Conference on Semantic Computing, August 4–7, 2008, Santa Clara, CA, USA*. IEEE, 2008.
9. Web ontology language. http://www.w3.org/2004/OWL/.
10. Resource description framework. http://www.w3.org/RDF/.
11. Kia Teymourian, Lyndon Nixon, Daniel Wutke, Reto Krummenacher, Hans Moritsch, Eva Khn, and Christian Schreiber. Implementation of a novel semantic web middleware approach based on triplespaces. In *Second IEEE International Conference on Semantc Computing, August 4–7, 2008, Santa Clara, CA, USA*, pages 518–523. IEEE, 2008.