Knowledge-Based Systems 22 (2009) 509–515

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Exploring design space for an integrated intelligent system

Nick Hawes *, Jeremy Wyatt, Aaron Sloman

Intelligent Robotics Lab, School of Computer Science, University of Birmingham, Birmingham B15 2TT, UK

ARTICLE INFO

Article history:
Available online 9 January 2009

Keywords:
Information-processing architectures
Intelligent systems
Robotics

ABSTRACT

Understanding the trade-offs available in the design space of intelligent systems is a major unaddressed element in the study of Artificial Intelligence. In this paper, we approach this problem in two ways. First, we discuss the development of our integrated robotic system in terms of its trajectory through design space. Second, we demonstrate the practical implications of architectural design decisions by using this system as an experimental platform for comparing behaviourally similar yet architecturally different systems. The results of this show that our system occupies a "sweet spot" in design space in terms of the cost of moving information between processing components.

1. Introduction

Intelligent systems (e.g. intelligent service robots) are a product of the many design decisions taken to ensure that the final system meets the requirements necessary to fit in its particular niche [19]. In nature, evolution creates behaviours and bodies that suit an animal's ecological niche. In the field of intelligent artifacts, choices about the design and implementation of hardware and software may be taken by a designer, or enforced by project or resource constraints. Few, if any, of these choices are truly independent; using a particular solution for one part of the system will constrain the space of solutions available for other parts of the system. For example, the number of degrees of freedom of an effector will restrict the design of the control software and behaviours required to use the effector, and the choice of middleware for software components will restrict the communication patterns that components can use. Understanding the *trade-offs available in the design space of intelligent artifacts* is a major open issue in the understanding of integrated intelligent systems, and thus AI.

In this paper, we focus on the design space of architectures for intelligent robots. We discuss the design of, and the trade-offs created by, an *architecture schema* for intelligent agents based on a model of shared working memories. Following this we present a novel exploration of the design space of information sharing models for architectures for integrated intelligent systems based on this schema. This exploration uses an intelligent robot as an experimental platform. The robot's architecture is varied in principled ways to generate quantitative information demonstrating the costs and benefits of the different designs.

* Corresponding author.
E-mail addresses: n.a.hawes@cs.bham.ac.uk (N. Hawes), j.l.wyatt@cs.bham.ac.uk (J. Wyatt), a.sloman@cs.bham.ac.uk (A. Sloman).
URLs: http://www.cs.bham.ac.uk/~nah (N. Hawes), http://www.cs.bham.ac.uk/~jlw (J. Wyatt), http://www.cs.bham.ac.uk/~axs (A. Sloman).

0950-7051/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2008.10.010

2. Background

In the field of intelligent systems, the term "architecture" is still used to refer to many different, yet closely related, aspects of a system's design and implementation. Underlying all of these notions is the idea of a collection of units of functionality, information (whether implicitly or explicitly represented) and methods for bringing these together. At this level of description there is no real difference between the study of architectures in AI and software architectures in other branches of computer science. However, differences appear as we specialise this description to produce architectures that integrate various types of functionality to produce intelligent systems. Architectures for intelligent systems typically include elements such as fixed representations, reasoning mechanisms, and functional or behavioural component groupings. Once such elements are introduced, the trade-offs between different designs become important. Such trade-offs include the costs of dividing a system up to fit into a particular architecture design, and the costs of using a particular representation. Such trade-offs have been ignored by previous work on integrated systems, yet these factors are directly related to the efficacy of applying an architecture design to a particular problem. Our research studies architectures for integrated, intelligent systems in order to inform the designers of these systems of the trade-offs available to them.

An important distinction to make when studying the information-processing architectures used in intelligent systems is the distinction between architectures that are entirely-specified in advance (e.g. those used in [16,17]), and architectures that are partially specified. This latter type can then be specialised to produce different *instantiations* of the architecture. We will refer to such partially specificed architectures as architecture *schemas* as they provide outlines from which many different concrete instantiations can be designed. Examples of such schemas include cognitive modelling architectures such as ICARUS [15], ACT-R [1], and Soar

[14]; more general frameworks such as CogAff [19] and APOC [2]; and robotic architectures such as 3T [6]. It is worth noting that the freedom in specialisation available to the designer varies greatly across these schemas.

As architecture schemas can be instantiated in various ways, each one provides a framework for exploring a limited region of design space for possible architecture instantiations: the use of a particular schema restricts the available design space to be those designs that can be created within the schema. Although instantiations produced from a schema may vary considerably, they will all share certain characteristics as a consequence of occupying similar regions of design space. It is difficult to study these characteristics directly, particularly in implemented systems, because it is difficult to separate the effects of the schema (i.e. the aspects of the architecture that will exist in all instantiations of the schema) from the effects of the specialisation (i.e. any additional components and implementation work) in the finished system.

As we wish to study the effects of variations in architecture schema in implemented systems we need an experimental approach that overcomes the problem of separating schema effects from specialisation effects. Our approach involves taking a single task (i.e. a problem requiring a fixed set of known behaviours) and creating instantiations of a number of different architecture schemas to solve it. In this way the task-specific elements of the instantiations are invariant (e.g. the algorithms used to process input and generate output), whilst the schema-level elements change between instantiations (e.g. the nature of the connections between input and output modules). Assuming task-specific invariance exists, comparing instantiations of different schemas on a single task will then provide information about the trade-offs between the different design options offered by the schemas.

Such single-task comparisons could be performed using existing systems. For example, driving tasks have been tackled in ACT-R [18] and ICARUS [5], spatial reasoning tasks by SOAR [20] and ACT-R [13], and natural language understanding has been tackled with almost every architecture schema, e.g. in APOC [3]. The drawback of this approach is that the different instantiations performed by different researchers using different technology will almost certainly introduce variations in behaviour that may mask the underlying effects of the various architectures, making comparisons worthless. To make comparisons between implemented systems informative, the variation in instantiations must be controlled. This is an approach we explore in Section 4. An alternative approach is to perform these comparisons theoretically (cf. [12]), although this risks overlooking the critical aspects that only become apparent when building integrated systems.

3. From requirements to robots

To further explore the idea of design decisions constraining the design space available for a particular intelligent system, it is worth considering an example. Our current research project is studying the problem of building intelligent systems. We are approaching the problem from various perspectives including hardware control, subsystem design (including vision, planning etc.) and architectures. As the project has progressed we have made strong commitments to particular designs for elements of the overall system. These design commitments have constrained the space of solutions available for subsequent developments. Although the following description is anecdotal, it demonstrates one possible type of development trajectory[1] for an integrated intelligent system.

[1] It is arguable that this development trajectory has much in common with a large number of intelligent and integrated system projects.

Prior to any design or development we analysed our target scenarios. From these scenarios we extracted a number of requirements for our integrated systems to satisfy, providing some initial constraints on design space. These requirements are too numerous to explore fully here, but the following proved important for subsequent developments:

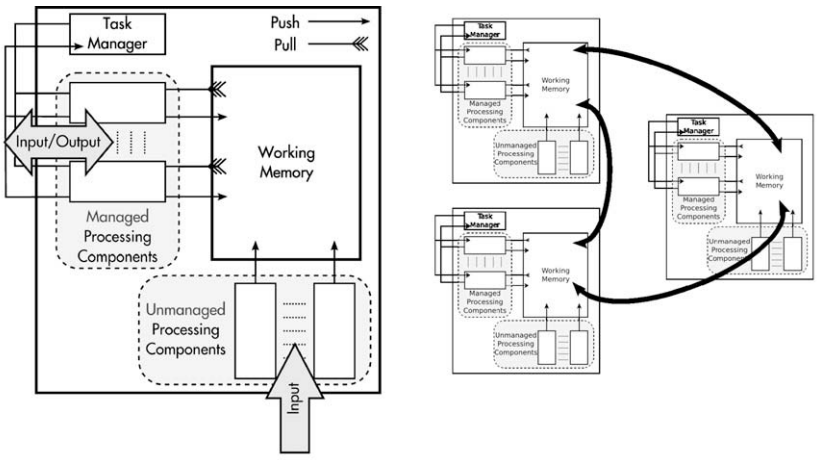
- The system must feature concurrently active elements in order to respond to a dynamic world whilst processing.
- The system must represent and reason about hypothetical future states, thus requiring explicit representations.
- The system must support specialised reasoning in its subsystems, requiring support for multiple representations.

Although these requirements do not appear too restrictive, they rule out design approaches that require a single unified representation and that do not support concurrency. This prevents the use of many architectures for modelling human-level intelligence, and logic-based robotics approaches.

Our first design for a system to satisfy the scenario's requirements was constructed using the Open Agent Architecture (OAA) [4]. It featured concurrent components for language interpretation and generation, object category recognition using a modified variant of SIFT, generation of multiple forms of spatial representations, and cross-modal information fusion. The use of OAA constrained us to design the system as a network of exhaustively pair-wise connected components that exchanged information directly. Although the resulting system satisfied our requirements and demonstrated the desired behaviour, the architecture structure had a number of drawbacks. The main drawback was that the direct exchange of information made it difficult to share the same information between more than two components in a system, making it difficult to explore the consequences of component collaboration (e.g. using scene information to incrementally reduce the hypothesis space during parsing [3]). This is a clear of example of a design choice (the connection model enforced by the architecture) limiting the subsequently available design space (the design space of information sharing models). It is worth noting that in theory we could have implemented a different information sharing model on top of OAA, but this would not have been a natural fit with the underlying architecture. These kinds of specialisation costs (i.e. the cost of implementing one system given the constraints of another) are hard to measure, but typically very important in the design and implementation of intelligent systems.

Because of the drawbacks of our OAA-based system, we decided to explore designs for architectures based on commonly accessible information stores. This led to the development of the CoSy Architecture Schema (CAS) [9,7,10], a schema that groups concurrently active processing components into separate *subarchitectures* via limited connections to *shared working memories*. In CAS all communication between components occurs via these working memories (rather than directly between components), enforcing the requirement of data sharing (see Fig. 1a). However, the separation of components into groups around working memories brings its own set of design constraints, and these have become apparent in the systems we have built on top of CAS.

The systems we have built with CAS feature subarchitectures designed around particular representations, e.g. a visual subarchitecture containing 3D representations of objects, and a communication subarchitecture containing logical interpretations of utterances. Although these representations are shared, they are only intelligible by the components in the same subarchitecture. To obtain a complete view of all the information in the system, our instantiations have had to include a *binding* subarchitecture, which abstracts from other subarchitectures to provide a single amodal representation of the current state [11]. It is arguable that



(a) The CAS subarchitecture design schema. All information passes via the working memory.

(b) A CAS architecture based on three subarchitectures. Cross-subarchitecture communication occurs via connections between working memories.

Fig. 1. Two views of the CoSy Architecture Schema.

a system based on a single unified data store would not require such augmentation (although such a design would not meet our requirements). This again demonstrates how a design decision (separating information is into different memories) influences the subsequent design choices for a system. In this instance, choices were not completely ruled out, but additional mechanisms were required to support functionality that would have been more easily obtained in an alternative design.

4. Exploring information sharing designs

The preceding discussion presents insights into system design in an entirely qualitative manner. To further the principled exploration of the design space of integrated intelligent systems, we want to generate quantitative data that describes possible trade-offs between designs. Given our experience with information sharing in different architectures, we chose to explore this area of design space using the comparative methodology described in Section 2.

There is a space of possible models for information sharing between components, ranging from point-to-point communication in which components share information only with directly connected components (i.e. the model used by the OAA-based system described in Section 3), to a broadcast model where components share information with every component in a system. Between these two extremes exist a range of possible systems in which components share information with a designer-defined subset of components. Which model is chosen can have a great impact on the behaviour of the final system, but little information is available to designers of intelligent systems about the trade-offs associated with this dimension of design space. In the remainder of this paper we make an initial attempt at filling this void.

4.1. Experimental system

The system which we have used to explore the effects of information sharing is derived from an intelligent robot cable of object manipulation and human-robot interaction [7]. It is based on the CAS architecture schema. This schema was briefly described in Section 3, and is pictured in Fig. 1. This schema has a number of features relevant to developing integrated systems, but here we will focus on the approach it takes to passing information between components[2].

When information is operated on (added, overwritten or deleted) in a CAS working memory a *change event* is generated. This event structure contains information on the operation performed, the type (i.e. class name) of the information, the component which made the change, and the location of the information in the system. Components use this change event data to decide whether to perform some processing task with the changed information. To restrict the change events that are received, each component is able to *filter* the event stream based on change event contents. Components typically subscribe to relevant change information by registering callbacks on combinations of the fields in the change event. For example, a vision component may subscribe to additions of regions of interests. This filter would refer to the change event's operation type (addition) and data type (region of interest).

CAS groups components into subarchitectures around a working memory. These subarchitecture groupings influence the flow of change events, and thus the flow of information between components. Within a subarchitecture components a sent all the of change events generated by operations on that subarchitecture's working memory. They then use their filters to select the relevant events from this stream. Change events that describe operations on other working memories (i.e. those outside of the subarchitecture) are first checked against the union of all of the filters registered by components in a subarchitecture. If an event passes these filters then it is forwarded to all of the subarchitecture's components via the same mechanism used for local changes. This reuse of the existing filter mechanism adds redundancy to the change propagation mechanisms, but reduces the complexity of the system.

When a component reads information from, or writes information to, a working memory, or a change event is broadcast, a *communication event* occurs. A communication event abstracts away from the underlying communications infrastructure, hiding

[2] For a more complete description of CAS, please see previous work, e.g. [9,7,10].

whether the information is being moved in memory, over a network or translated between programming languages. Within subarchitectures any operation requires a single communication event. When communication happens between two subarchitectures an additional communication event is required due to the separation (this is equivalent to the information passing over one of the dark lines in Fig. 1 (b)).

Change and communication events are implemented as part of our CoSy Architecture Schema Tookit (CAST) which realises the CAS schema in an open source, multi-language software framework [10]. In CAST the change event system is implemented as a callback mechanism modelled on event driven programming. The communication events are implemented as procedure calls or remote procedure calls depending on the languages and distribution methods used in the instantiation.

Using CAST we have built an integrated intelligent robot which featuring subarchitectures for vision, qualitative spatial reasoning (QSR), communication, continual planning, binding, manipulation and control [7]. To provide a simpler system useful for exploring the design space of information sharing mechanisms, reduced this system to a smaller number of subarchitectures: vision, binding, and QSR. This reduction was chosen because it provides a simpler system which still integrates two modalities with distinct representations (quantitative vision and qualitative spatial reasoning). For the experimental runs we replaced some of the components in the visual subarchitecture with simulated components. These not only simulated the results of visual processing, but also the interactions of the components via shared working memories (the important aspect of this study). This allowed us to fully automate interactions with the system in order to perform a large number of experimental runs. Aside from these alterations, the remaining components were taken directly from our original robotic system.

When presented with an object after a change to the visual scene, the system first determines its 3D position and then extracts some visual attributes. This information is abstracted into the binding subarchitecture where it become available in a simplified form to the rest of the system. The presence of object information in the binding subarchitecture triggers the QSR subarchitecture which computes spatial relations between the new object and any other known objects. This relational information is transmitted back to the binding subarchitecture where the relations are introduced between the existing object representations.

4.2. Methodology

We can use the shared memory-based design of CAS to explore the effects of varying information sharing patterns between components in our experimental system. We do this by altering the ratio of components to subarchitectures.

We start with an $n:m$ design where n components are divided between m subarchitectures, where $n > m > 1$. This is our original system described above, in which components are assigned to subarchitectures based on functionality (vision, binding or QSR), although for this experimental work arbitrary $n{:}m$ assignments are also possible (and would explore a wider area of design space). We then reconfigure this system to generate architectures at two extremes of the design space for information sharing models. At one extreme we have an $n{:}1$ design in which all n components from the original system are in the same subarchitecture. At the other extreme of design space we have an $n{:}n$ design in which every component is in a subarchitecture of its own. Each of these designs can be considered a schema specialisation of the CAS schema from which a full instantiation can be made.

These various designs are intended to approximate, within the constraints of CAS, various possible designs used by existing systems. The $n{:}1$ design represents systems with a single shared data store to which all components have the same access. The $n{:}m$ design represents systems in which a designer has imposed some modularity which limits how data is shared between components. The $n{:}n$ design represents a system in which a no data is shared, but is instead transmitted directly between components. In the first two designs a component has do to extra work to determine what information it requires from the available shared information. In the latter two designs a component must do extra work to obtain information that is not immediately available to it (i.e. information that is not in its subarchitecture's working memory).

In order to isolate the effects of the architectural alterations from the other runtime behaviours of the resulting systems, it is important that these architectural differences are the only differences that exist between the final CAS instantiations. It is critical that the systems are compared on the same task using the same components. CAST was designed to support this kind of experimentation: it allows the structure of instantiations to be changed considerably, with few, if any, changes to component code. This has allowed us to take the original implementation described above and create the $n{:}1$, $n{:}m$, and $n{:}n$ instantiations without changing component code. This means that we can satisfy our original aim of comparing near-identical systems on the same tasks, with the only variations between them being architectural ones.

To measure the effects of the architecture variations, we require metrics that can be used to highlight these effects. We previously presented a list of possible metrics that could be recorded in an implemented CAS system to demonstrate the trade-offs in design space [8]. Ultimately we are interested in measuring how changes to the way information is shared impacts on the external behaviour of the systems, e.g. how often it successfully completes a task. However, given the limited functionality of our experimental system, these kind of behaviour metrics are relatively uninformative. Instead we have chosen to focus on lower-level properties of the system. We have compared the systems on

(1) Variations in the number of filtering operations needed to obtain the change events necessary to get information to components as required by the task.
(2) Variations in the number of communication events required to move information around the system.

As discussed previously communication and change events underlie the behaviour of almost all of the processing performed by a system. Therefore changes in these metrics demonstrate how moving through the space of information sharing models supported by CAS influences the information processing profile of implemented systems.

We studied the three different designs in two configurations: one with vision and binding subarchitectures, and the second with these plus the addition of the QSR subarchitecture. This resulted in six final instantiations which we tested on three different simulated scenes: scenes containing one object, two objects and three objects. Each instantiation was run twenty times on each scene to account for variations unrelated to the system's design and implementation.

4.3. Results

The results for the filtering metric are based around the notion of a *relevant event*. A relevant event is a change event that a component is filtering for (i.e. an event that it has subscribed to). Fig. 2 demonstrates the percentage of relevant events received per component in each instantiation. 100% means that a component only receives change events it is listening for. A lower percentage means that the connectivity of the system allows more than the relevant

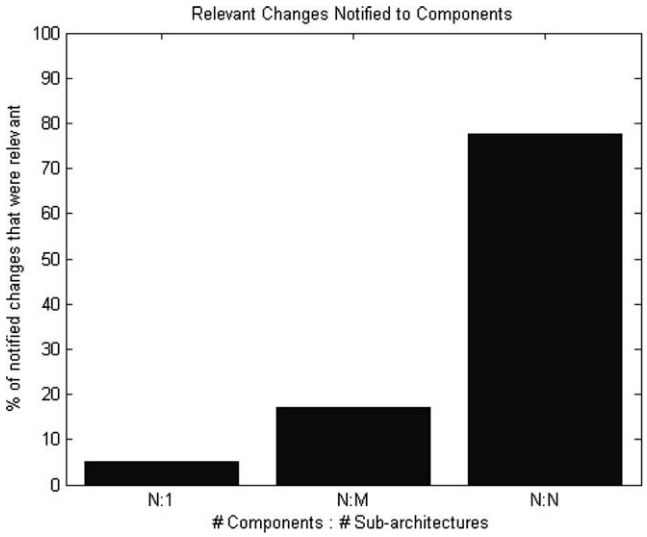


Fig. 2. Average number of relevant change events received per component.

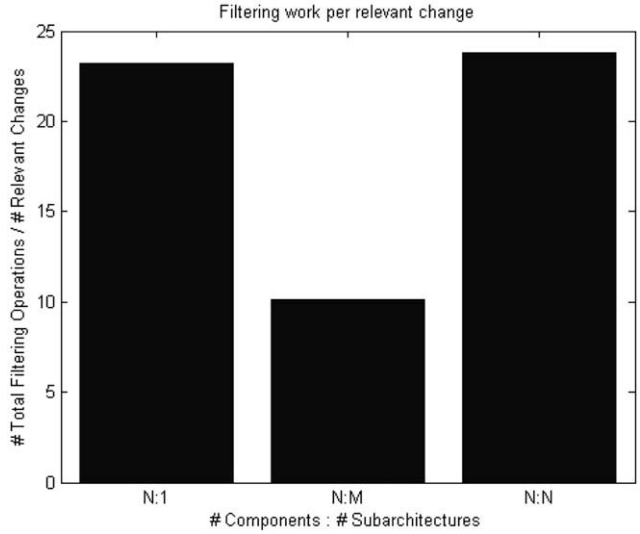


Fig. 3. Average filtering effort per relevant change event received.

change events to get the component, which then has to filter out the relevant ones. This is perfectly natural in a shared memory system. The results demonstrate that a component in an $n:1$ instantiation receives the lowest percentage of relevant events. This is because within a subarchitecture, all changes are broadcast to all components, requiring each component to do a lot of filtering work. A component in an $n:n$ instantiation receives the greatest percentage of relevant changes. This is because each component is shielded by a subarchitecture working memory that only allows change events that are relevant to the attached components to pass. In the $n:n$ case because only a single component is in each subarchitecture this number is predictably high.[3] This figure demonstrates the benefits of a directly connected instantiation: components only receive the information they need.

However, this increase in the percentage of relevant changes received comes at a cost. If we factor in the filtering operations being performed at a subarchitecture level (which could be considered as "routing" operations), we can produce a figure demonstrating the total number of filtering operations (i.e. both those at a subarchitecture and a component level) per relevant change received. This is presented in Fig. 3. This shows a striking similarity between the results for the $n:1$ and $n:n$ instantiations, both of which require a larger number of filtering operations per relevant change than the $n:m$ instantiations. In the $n:m$ systems, the arrangement of components into functionally themed subarchitectures results in both smaller numbers of change events being broadcast within subarchitectures (because there are fewer components in each one), and a smaller number of change events being broadcast outside of subarchitectures (because the functional grouping means that some changes are only required within particular subarchitectures). These facts mean that an individual component in an $n:m$ instantiation receives fewer irrelevant change events that must be rejected by its filter. Conversely a component in the other instantiations must filter relevant changes from a stream of changes containing all of the change events in the system. In the $n:1$ instantiations this is because all of these changes are broadcast within a subarchitecture. In the $n:n$ instantiations this is because all of these changes are broadcast between subarchitectures. Fig. 4 shows that these results are robust against changes in the number of objects in a scene. Also, the nature of the results did not change between the systems with vision and binding components, and those with the additional QSR components.

Fig. 5 demonstrates the average number of communication events per system run across the various scenes and configurations for the three different connectivity instantiations. This shows that an $n:n$ instantiation requires approximately 4000 more communication events on average to perform the same task as the $n:1$ instantiation, which itself requires approximately 2000 more communication events than the $n:m$ instantiation. Fig. 6 demonstrates that this result is robust in the face of changes to the number of objects in a scene. The nature of the results also did not change between the systems with vision and binding components, and those with the additional QSR components.

This result is due to two properties of the systems. In the $n:n$ system, every interaction between a component a working memory (whether it is an operation on information or the propagation of a change event) requires an additional communication event. This is because all components are separated by subarchitectures as well as working memories. In addition to this, the number of change events propagated through the systems greatly effect the

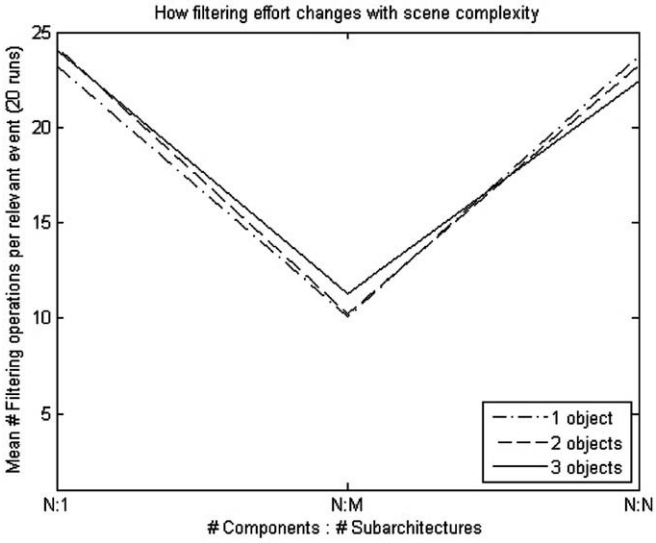


Fig. 4. Average filtering effort per relevant event compared to scene complexity.

[3] The events required by the manager component in each subarchitecture mean the relevant percentage for the *n*-*n* instantiations is not 100%.

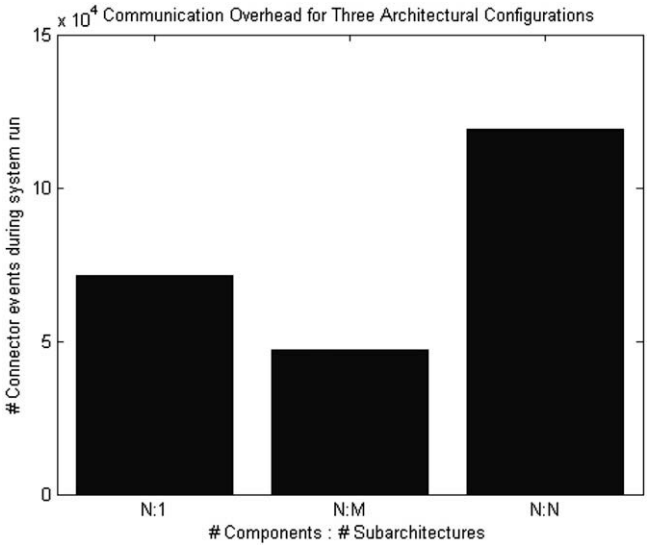


Fig. 5. Average total communication events per instantiation run.

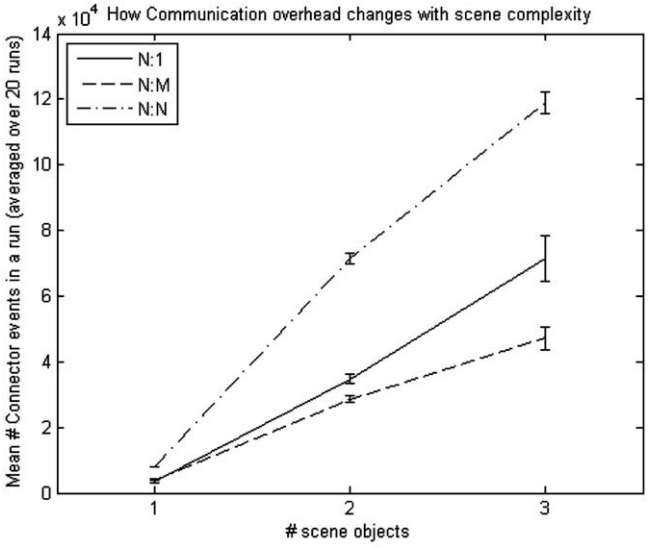


Fig. 6. Average total communication events per instantiation run compared to scene complexity.

amount of communication events that occur. In the $n{:}n$ and $n{:}1$ instantiations, the fact that they effectively broadcast all change events throughout the system contributes significantly to the communication overhead of the system.

5. Conclusions

From these results we can conclude that a functionally-decomposed $n{:}m$ CAS instantiation occupies a "sweet spot" in architectural design space with reference to filtering and communication costs. This sweet spot occurs because having too much information shared between components in a system (the $n{:}1$ extreme) means that all components incur an overhead associated with filtering out relevant information from the irrelevant information. At the other extreme, when information is not shared by default (the $n{:}n$ extreme) there are extra communication costs due to duplicated transmissions between pairs of components, and (in CAS-derived systems at least) the "routing" overhead of transmitting information to the correct components (i.e. the filtering performed by working memories rather than components).

In this simple example the existence of such a sweet spot, subject to well defined assumptions, could be established mathematically without doing any of these experiments. However, we have shown the possibility of running experiments to test such mathematical derivations, and also to deal with cases where no obvious mathematical analysis is available because of the particular features of an implementation.

It is clear that our results are not the end of the story. We have yet to explore $n{:}m$ instantiations that are not designed along functional lines; it seems sensible to expect them not to perform as well as the $n{:}m$ system presented here. It is also not clear that $n{:}n$ instantiations in CAS accurately capture the benefits of a directly connected system, as CAS's design is tailored to information sharing as a default. This observation leads us to consider an open question: what other appropriate metrics should be considered when evaluating trajectories through design space? In this paper we considered relatively low-level metrics because they could be captured and characterised relatively easily. Other relevant metrics include behavioural measures (e.g. how likely a system is to achieve a goal), expressiveness measures (e.g. how easy it is to encode a particular solution to a problem in an architecture), and meta-level measures (e.g. how easy it is for a designer, or the system itself, to reconfigure the architecture or alter its functionality). It is only when this whole space of possibilities is addressed can we truly start to judge the trade-offs of designing an architecture in a particular way (with reference to a particular task).

Even given these shortcomings, the novel experimental methodology presented in this paper points to a route forward for the principled study of integrated intelligent systems in AI. It is our hope that further work along these lines will provide system designers with a body of knowledge about the choices and trade-offs available in architectural design space, allowing them to build systems that satisfy their requirements in an informed and principled manner.

Acknowledgement

This work was supported by the EU FP6 IST Cognitive Systems Integrated Project "CoSy" FP6-004250-IP, and the EU FP7 IST Cognitive Systems Integrated Project "CogX" ICT-215181-CogX.

References

[1] J.R. Anderson, D. Bothell, M.D. Byrne, S. Douglass, C. Lebiere, Y. Qin, An integrated theory of the mind, Psychological Review 111 (4) (2004) 1036–1060.
[2] V. Andronache, M. Scheutz, Integrating theory and practice: the agent architecture framework apoc and its development environment ade, in: AAMAS'04, 2004, pp. 1014–1021.
[3] T. Brick, M. Scheutz, Incremental natural language processing for hri, in: HRI'07, 2007, pp. 263–270.
[4] A. Cheyer, D. Martin, The open agent architecture, Autonomous Agents and Multi-Agent Systems 4 (1) (2001) 143–148.
[5] D. Choi, M. Morgan, C. Park, P. Langley, A testbed for evaluation of architectures for physical agents, in: AAAI'07 WS: Evaluating Architectures for Intelligence, 2007.
[6] E. Gat, On three-layer architectures, in: D. Kortenkamp, R.P. Bonnasso, R. Murphy, (Eds.), Artificial Intelligence and Mobile Robots, 1997.
[7] N. Hawes, A. Sloman, J. Wyatt, M. Zillich, H. Jacobsson, G.-J. Kruijff, M. Brenner, G. Berginc, D. Skočaj, Towards an integrated robot with multiple cognitive functions, in: AAAI'07, 2007b, pp. 1548 –1553.
[8] N. Hawes, A. Sloman, J. Wyatt, Towards an empirical exploration of design space, in: Proceedings of the 2007 AAAI Workshop on Evaluating Architectures for Intelligence, 2007a, Vancouver, Canada.
[9] N. Hawes, J. Wyatt, A. Sloman, An architecture schema for embodied cognitive systems, in: Technical Report CSR-06-12, University of Birmingham, School of Computer Science, 2006.
[10] N. Hawes, M. Zillich, J. Wyatt, BALT & CAST: middleware for cognitive robotics, in: Proceedings of IEEE RO-MAN, 2007, pp. 998–1003.
[11] H. Jacobsson, N. Hawes, G.-J. Kruijff, J. Wyatt, Crossmodal content binding in information-processing architectures, in: Proceedings of the Third ACM/IEEE International Conference on Human-Robot Interaction (HRI), Amsterdam, The Netherlands, March 12–15, 2008.

[12] R.M. Jones, R.E. Wray, Comparative analysis of frameworks for knowledge-intensive intelligent agents, AI Magazine 27 (2) (2006) 57–70.
[13] W.G. Kennedy, M.D. Bugajska, M. Marge, W. Adams, B.R. Fransen, D. Perzanowski, A.C. Schultz, J.G. Trafton, Spatial representation and reasoning for human-robot collaboration, in: AAAI'07, 2007, pp. 1554–1559.
[14] J.E. Laird, A. Newell, P.S. Rosenbloom, Soar: an architecture for general intelligence, AIJ 33 (3) (1987) 1–64.
[15] P. Langley, D. Choi, A unified cognitive architecture for physical agents, in: AAAI'06, 2006.
[16] N. Mavridis, D. Roy, Grounded situation models for robots: where words and percepts meet, in: IROS'06, 2006.
[17] P. Mcguire, J. Fritsch, J.J. Steil, F. Rothling, G.A. Fink, S. Wachsmuth, G. Sagerer, H. Ritter, Multi-modal human-machine communication for instructing robot grasping tasks, in: IROS'02, 2002.
[18] D.D. Salvucci, Modeling driver behavior in a cognitive architecture, Human Factors (48) (2006) 362–380.
[19] A. Sloman, The "semantics" of evolution: trajectories and trade-offs in design space and niche space, in: IBERAMIA'98, 1998, pp. 27–38.
[20] S. Wintermute, J.E. Laird, Predicate projection in a bimodal spatial reasoning system, in: AAAI'07, 2007, pp. 1572–1577.