A Structural Synthesis System for LCC Protocols

Argyrios Grivas (0455134)

March 14, 2005

1 Introduction

The LCC (Lightweight Communication Calculus) language is a promising approach for decentralized coordination of Multi-Agent Systems (MAS) in open environments. LCC is used for the specification of interaction protocols, which can be executed by the agents. So far, there is no tool for the structured design of such LCC protocols. The designers are supposed to write the protocols nearly from scratch and without any kind of assistance or consistency check. The purpose of the proposed work is to develop a tool for the structured design of LCC protocols. We expect that the LCC approach to MAS coordination is going to benefit from the existence of such a tool since the specification of protocols will be substantially improved in terms of speed and correctness.

2 Motivation

One of the most promising approaches for MAS coordination is LCC. LCC is an executable specification language in the sense that mechanisms exist for deploying LCC protocols when coordinating software components. Although the LCC approach has a strong theoretical basis and seems to provide a good balance between efficient coordination and respect of the agents' independence, there is some work to be done yet from the software engineering point of view. In particular, one of the potential issues that should be addressed is the structured design of LCC specifications. At the moment, the designers have to write protocols by hand and no kind of tool that supports this process in any way exists. This observation will the basic motivation for the current proposal.

The aim of the proposed project is to develop a structured design editor for LCC protocols. In order to achieve this goal, a closer look to LCC itself is required. LCC is basically a process calculus, but has a lot in common with Horn clause specification. Particularly, the pre and post conditions, which actually determine the circumstances under which the messages can be sent, have a lot in common with logic programming. It is therefore straightforward to consider using structured design techniques from logic programming in LCC specifications. Techniques editing [2] is a method used in Prolog, which seems to adapt well up to a certain point to the LCC case. On the other hand, the fact that LCC is a process calculus allows us to consider the use of process oriented design techniques. A good combination probably is to use process-oriented methods to obtain the "skeletons" used by Techniques editing.
Apart from the practical value of the tool, there is also an interesting research aspect. By the development of the editor we will be testing the following hypothesis: There are existing patterns, which can be used for building LCC protocols, and the use of such patterns has several advantages in terms of usability. The hypothesis can be split into two parts. The first part states that suitable patterns already exist and can adapt to the LCC case. The second part claims that the use of such a pattern-based technique in a design tool for protocols will have a positive impact to the usability of the tool. In order to test the first one we are going to examine how existing techniques and particularly Techniques editing is going to adapt to the specific problem. The development of the editor will be based on the outcome of this effort and its main purpose is to demonstrate the resulting method. The second part of the hypothesis is going to be evaluated only if there is enough time to experiment on the usability of the tool with real users.

The reason why such an approach would be useful today, is basically its role in MAS development. The coordination of large MAS in open environments is an extremely interesting subject for the agents’ research community and LCC is a promising approach for tackling this problem. Given the significance of LCC and the fact that its theoretical basis is already sufficient, a tool that contributes to the software engineering aspect of the approach is considered to be a valuable advance for the whole method.

Another interesting aspect of the proposed work is feasibility. We have already argued that existing structured editing techniques seem to apply to the LCC case. The similarity of LCC with both logic programming and process-oriented methods provides us with a initial set of techniques which are common in these two fields. Techniques editing is a methodology used in logic programming which seems to be the most suitable for LCC protocols. In this approach a logic program is constructed by applying a set of syntactic transformations called techniques to a basic structure of the program, which is called skeleton [8]. An adaptation of this approach for specifying protocols in LCC is to obtain the skeleton using process-oriented methods and then refine the design using techniques used in logic programming. Although this is a good start for our research, it is not yet clear how well these techniques are going to adapt to protocol specification and this is the main question that our research will attempt to answer.

Finally, a major part of the motivation for the proposed research project is its significance for the research community and the novelty beneficiaries that its success will introduce. We have argued for the significance of MAS for the agent community. A known issue not only for LCC but for several approaches for the development of large MAS in open environments, is the lack of tools and methods regarding the software engineering point of view. Our attempt is going to introduce the use of software engineering techniques in MAS design and therefore it is considered a significant advance for the development of MAS. The fact that the structured design editor is supposed to make protocol design quicker, easier and more accessible to engineers not extremely familiar with LCC, is the basically the novelty introduced by the proposed project. Immediate consequence of this is that the whole Electronic Institutions methodology, the most popular approach for developing large MAS and a major field where LCC would be very useful, will be much more accessible to MAS engineers.
3 Background material

3.1 LCC

LCC (Lightweight Communication Calculus) is a language for specifying interaction protocols for multi-agent systems. It is an executable specification language in the sense that mechanisms exist for deploying LCC protocols when coordinating software components. A formal definition of LCC is provided in [4]. One aspect of the effort, as it is stated by the author, is the invention of a logic programming language that provides an overall architecture for the coordination of multi-agent systems. The whole architecture is described by showing how the main issues of developing MASs are tackled using techniques taken from programming. In particular, it is explained how:

- the interaction model can be represented in process calculus,
- the social norms that define the message passing behavior of the agents can be applied by the satisfaction of mutual constraints individually to the agents,
- the state change of the interaction maps to the unfolding of a clause with respect to the protocol,
- the expansion of the distributed clause coordinate the agents,
- the interaction scope corresponds to constraints applied to the values that the variables can take and
- the brokering can be achieved by providing additional information for the agents (i.e. capability descriptions)

LCC approach to MAS coordination has at least two major advantages over the Electronic Institutions (EI) approach as it is described in [7]. The first one is the completely decentralized coordination of the MAS. In LCC the execution of the protocol itself is adequate to coordinate the interaction between the agents. It is quite obvious that this is a key benefit especially for the case of open environments. The second major advantage that makes LCC a promising approach, is the minimal requirements for the architecture of the agent. It is usually the case in open environments that no assumptions can be made about the internal structure of the agents, since they have been separately developed by different engineers using different approaches. It is clear that any coordination mechanism for such environments has to make as few as possible assumptions about the participants and therefore LCC seems to adapt very well in that case. To sum up, LCC seems to tackle some of the problems of the coordination in open environment in a more efficient way than the popular EI approach and therefore it is definitely worth the attention of the MAS community.

3.2 Techniques editing

A programming technique is, as it is described in [6], a common to the programmers code pattern, specific to a particular language, but irrelevant to the algorithm or the problem domain, which is regularly used. The use of such techniques is very common in logic programming. Expert users in Prolog are
aware of constructs like the ”accumulator pair” and use them regularly. A good example, taken from [2], is to consider the implementation of ”quick” reverse in Prolog:

\begin{verbatim}
rev([],R,R)
rev([H|T],R0,R) :-
    rev(T,[H|R0],R).
\end{verbatim}

The predicate above consists of two parts. The first one performs the recursion down the list

\begin{verbatim}
rev([],...)
rev([H|T],...) :-
    rev(...).
\end{verbatim}

and the second one builds a list during the recursion (accumulator pair)

\begin{verbatim}
rev([],R,R)
rev(...,R0,R) :-
    rev(...,[H|R0],R).
\end{verbatim}

These two parts are Prolog techniques in the sense that they are common code patterns used in wide variety of situations regardless the algorithm being implemented.

A methodology for constructing logic programs using techniques is proposed in [5]. The key points of the method, as they are summarized in [2] are the following:

- The construction of the program is based on the skeleton, which determines the control flow of the program.
- There is set of methods for performing simple tasks. These methods are called additions.
- Additions can be applied to the initial skeleton in order to obtain an extension.
- Extensions can be composed to produce fully functional logic programs.

One of the most interesting applications of this methodology is in Prolog editors (Techniques editing). Several editors using techniques have been developed. The approach described in [3] follows the methodology described above. The user is given a set of predefined skeletons and techniques. The program is constructed by applying different combinations of techniques on the initially selected skeleton. Despite the limitations regarding the library of techniques the editor described in [3] is a valuable implementation of the techniques editing approach. Another interesting techniques-based editor named Ted is described in [1]. Techniques in Ted are defined as relationships between the head and recursive arguments in the recursive clauses of a program. Even though it is different from the skeleton-addition approach, Ted demonstrates that techniques can be useful for teaching Prolog to novice users. The study of both the editors suggests that a techniques-based approach to editing has several advantages and should be applied in cases when a sufficient set of skeletons and additions can be defined.
4 Evaluation of hypothesis

The first part of the evaluation is the development of the editor. The implementation of this tool is going support the part of hypothesis claiming that there are existing methods which can adapt to the problem. The evaluation of the same part of the hypothesis will be finalized by testing the tool on benchmark examples of existing LCC protocols. The claim that a structured design editor will have a positive impact on usability, will be evaluated experimentally only in case there is sufficient time for an experiment with real users.

5 Methods and techniques

There are two parts of the proposed work, for which further discussion from the practical point of view is needed. The first one is the development of the prototypical techniques editor of LCC protocols. The editor will be a Java Swing application capable of saving the protocols in an XML representation compatible with the Java-based agent framework for LCC. The use of Java as the language was decided to simplify the future integration of the tool with the rest of the LCC framework and to enable the use of the editor in a variety of applications of LCC.

Another issue is the experimental evaluation of the structured editor. We should note that this evaluation will take place only if there is enough time left after the successful achievement of the primary goals. The participants are going to be real users of LCC with variable experience in defining protocols. The users will be asked to compose a given set of LCC protocols by using the current process of doing so and by using the techniques editor. The time needed to compose the protocols, the number of errors will be recorded in both cases and a statistical analysis will be applied to the result. Finally, apart from the result in summary, the impact of the user’s experience on the performance improvement using the tool would be an interesting parameter to evaluate.

6 Expected outcome

The main outcome of the project is the implemented structured design editor. Given that the effort is going to be successful, the editor has practical value since it can be integrated with the rest of the LCC framework and make the whole approach much more accessible to MAS engineers. Another valuable result will be the understanding of the suitability of techniques editing for languages like LCC and in particular for process oriented languages. The usability evaluation, if performed, will also provide evidence for the impact of such techniques on usability both for expert and novice users. Finally, the structured description of the range of protocols that the editor will be able to construct would also be a worthwhile result.
7 Research plan

7.1 Work packages and durations
The project is expected to be split into the following work packages:

1. Scenario development, taking existing LCC definitions and developing methods that apply to them. Then generalizing from these. **Duration:** 2 weeks

2. Design of an incremental method for specifying processes. **Duration:** 2 weeks

3. Design of a (possibly) techniques-based method for completing LCC definitions, starting from process skeletons. **Duration:** 2 weeks

4. Build a prototypical support tool that makes the methods above easy to apply. **Duration:** 4 weeks

5. Compare the method and tool to comparable tools for software specification. **Duration:** 1 week

6. Evaluation of the usability of the editor. **Duration:** 2 weeks

7. Composing of the final written report. **Duration:** 1 week

7.2 Deliverables and dependencies
The deliverables of the first three work packages will be on the form of a report summarizing the outcome of the relevant research. The editor is considered to be the deliverable of the forth work package, a report comparing the editor with existing tools will be the deliverable of the fifth, a report summarizing the experiment and its outcome is expected from the sixth and finally a large written report about the project is the deliverable of the last work package. The first four packages must be done sequentially in the given order. The last package will begin from the last week of the forth package. The fifth and the sixth package can be done simultaneously immediately after the forth one is completed.

7.3 Risks and backup plan
One of the risks, which has already been pointed out, is the lack of time to experiment on usability. Although we seem to have a good idea of how the project will run, it is not clear how "easily" the previously described methods are going to adapt to the LCC case and therefore more time than the current estimation might be required. The backup plan in that case is not to include the experiment to the final outcome of the project. We believe that this is not a major problem since the usability testing is not a primary goal for the specific project.

Another risk is the possibility that the method and the editor fail to support the full range of LCC protocols. This is not unlikely to happen, since it is not clear how suitable the existing methods are, for LCC specifications. The backup plan is to focus on the set of protocols which are considered to be the
most important. Although the risk is dangerous for the success of the project, we believe that the probability of failing to support any interesting set of protocols is very low.

References


